5-2006

Reconciling Gene Trees with Apparent Polytomies

Wen-Chieh Chang
Iowa State University, wcchang@iastate.edu

Oliver Eulenstein
Iowa State University, oeulenst@iastate.edu

Reconciling Gene Trees with Apparent Polytomies

Abstract
We consider the problem of reconciling gene trees with a species tree based on the widely accepted Gene Duplication model from Goodman *et al*. Current algorithms that solve this problem handle only binary gene trees or interpret polytomies in the gene tree as true. While in practice polytomies occur frequently, they are typically not true. Most polytomies represent unresolved evolutionary relationships. In this case a polytomy is called *apparent*. In this work, we modify the problem of reconciling gene and species trees by interpreting polytomies to be apparent, based on a natural extension of the Gene Duplication model. We further provide polynomial time algorithms to solve this modified problem.

Keywords
algorithm, computational biology, gene duplication, gene/species tree reconciliation

Disciplines
Theory and Algorithms

Reconciling Gene Tree with Apparent Polytomies

Wen-Chieh Chang and Oliver Eulenstein

TR #
August 2005, Revised March, May 2006

Keywords: Algorithm, computational biology, gene duplication, gene/species tree reconciliation.
1999 CR Categories: Life and Medical Sciences; Nonnumerical Algorithms and Problems;

Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1041, USA

Gene tree reconciliation with soft multifurcations

by

Wen-Chieh Chang

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Oliver Eulenstein, Major Professor
Drena Dobbs
David Fernández-Baca

Iowa State University
Ames, Iowa
2006

Graduate College
Iowa State University

This is to certify that the master's thesis of
Wen-Chieh Chang
has met the thesis requirements of Iowa State University

~~Major Professor~~

~~For the Major Program~~

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We consider the problem of reconciling gene trees with a species tree based on the widely accepted Gene Duplication model from Goodman *et al.* Current algorithms that solve this problem handle only binary gene trees or interpret polytomies in the gene tree as true. While in practice polytomies occur frequently, they are typically not true. Most polytomies represent unresolved evolutionary relationships. In this case a polytomy is called *apparent*. In this work, we modify the problem of reconciling gene trees by interpreting polytomies to be apparent, based on a natural extension of the Gene Duplication model. We further provide polynomial time algorithms to solve this modified problem.

1 INTRODUCTION

In order to predict the function of genes it is critical to distinguish between speciation and duplication events in the genes' common evolutionary history [18, 9]. Duplication events, which are pervasive in many gene families, Typically result in incongruence between evolutionary histories of genes and the histories of the species from which the genes were sampled from. Evolutionary histories of either genes or species are represented through rooted phylogenetic trees (where every internal node has at least two children) and we refer to them as either *gene* or *species trees* respectively. An example for incongruence between a gene and its species tree that is caused by gene duplication is depicted in Figure 1.1.

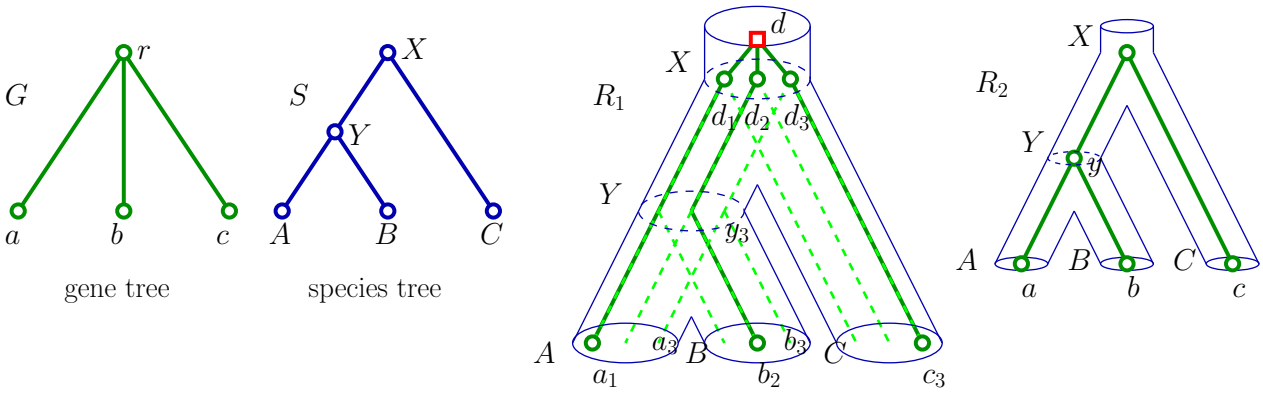

Figure 1.1 The gene and species tree are incongruent, assuming that the polytomies r in G is true. In our case the incongruence is caused by a duplication of the gene d into the copies d_1,d_2,d_3 that takes place in the root X of the species tree. The copies evolve along the topology of the species tree S and from the reconciled (gene) tree $R1$. Thus, inferring the gene tree from the genes a, b, c results in the gene tree G. Hence, the tree R_1 explains the incompatibility through the duplication d, If we interpret the polytomy r in G as apparent the unknown topology of G can be the topology of S. In this case, the reconciled (gene) tree R_2 has no duplication events and is equal to the gene tree G.

Background and Motivation

The gene duplication (GD) model from Goodman *et al.* [10] infers gene duplication events and losses from the incongruence of a given gene and species tree. While the basic GD model has been widely accepted and utilized through efficient algorithms [17, 6, 7, 14, 23], the model is constrained by its interpretation of nodes that have more then two children called polytomies. Polytomies can be either 'true' or 'apparent' [15, 21]. A polytomy is true if all of its children diverged from it at the same time. A polytomy is apparent when it replaces some phylogenetic subtree that could not be fully resolved in the evolutionary history. In practice, most gene trees contain numerous weakly supported or completely unresolved evolutionary relationships that may be represented most accurately by apparent polytomies. The basic GD model is constrained to true polytomies. Since true polytomies are rare evolutionary events, available algorithms for the GD model were mostly designed for only fully binary input trees.

The presented work introduces the first algorithm that infers gene duplications and losses from gene trees by interpreting polytomies as apparent. In this work, we (i) show a natural modification of the basic GD model for apparent polytomies, (ii) formulate the Reconciliation problem that infers duplications and losses from the modified GD model, (iii) present an overview of structural properties of the extended GD model, and (vi) derive from these properties a polynomial time algorithm that solves the reconciliation problem.

Previous work

Goodman *et al.* [10] introduced the GD model to infer gene duplication and losses for a given gene and species tree. This work was formalized and later refined by Page [17] and others [11, 16, 7, 4].

Given a gene and a species tree, the GD model assumes that a surjective mapping from the leaves of the gene tree to the leaves of the species tree is provided, that maps each gene to the species that it was sampled from. The idea of the GD model is theoretically to infer all possible gene trees, called *duplication trees*, from the species tree by allowing genes to either duplicate in two or more copies or to speciate. Duplication can only take place in one species and thus the duplicated gene and its copies have the same species. For example in Figure 1.1 in species X, gene d is duplicated into three copies d_1, d_2 and d_3. The copies evolve through further duplication or speciation along the topology of the subtrees of the species tree rooted at species X. Speciation occurs when a gene in a species evolves into gene for each child of the species. For example in Figure 1.1 gene y_3 in species Y speciates into the genes a_3 and b_3. Some of the duplication trees are evolutionary compatible with the gene tree. In this case, we can embed the gene tree into the duplication tree such that (i) each leaf in the gene tree and its embedding in the duplication tree have the same species and (ii) the least common ancestor relationships of the gene tree are preserved through the embedding. Figure 1.1 depicts an example where gene tree G can be embedded into the duplication tree R_1. Duplication trees that allow such an embedding of the gene tree are called *explanation trees*. Explanation trees are evolutionary compatible with the gene tree and thus explain the incompatibility between the gene tree and the species tree through gene duplication and losses. A *reconciled* (gene) tree is an explanation tree with the minimum number of nodes [17]. It was shown [7] that a reconciled tree is uniquely determined by its gene and species tree.

The *reconciliation cost* measures the fit between a gene and a species tree with respect to gene duplication. In a simplest way, the reconciliation cost is the number of duplication events in an

explanation tree. This measure may be refined by additionally counting the number of losses in the explanation tree. A *loss* is defined through the embedding of the gene tree into the explanation tree as a maximal subtree in the explanation tree that has no embedding from the gene tree.

As an example in Figure 1.1 the subtree rooted at node z_3 of the reconciled tree R_1 is a loss. Both measures can be computed in polynomial time in the size of the given trees [11, 7]. Alternatively, a reconciled tree can be defined as an explanation tree that minimizes the number of gene-duplications and losses. Under the basic GD model these definitions are equivalent [3].

The reconciled tree and its reconciliation cost can be computed in polynomial time from fully binary gene and species trees [17, 14, 6] or from gene and species trees when the polytomies are assumed to be true [7].

Presented work

In this work, we modify the basic GD model to interpret polytomies in gene trees as apparent. Apparent polytomies represent unknown phylogenetic subtrees. Thus, the idea of the modified GD model is to replace the polytomies by subtrees such that the nodes in the resulting reconciled tree are minimized. Therefore we consider the set $\mathcal{G}$ of all gene trees that we construct from the given gene tree G by replacing star trees, which are the polytomies and their children, with more refined trees. Let $\mathsf{Exp}_{G,S}$ be the set that contains the explanation trees for each combination of a gene tree in $\mathcal{G}$ and the species tree S in the basic GD model. Equivalently $\mathsf{Exp}_{G,S}$ can be described as the duplication trees into which the gene tree G can be embedded using a relaxed embedding function. The relaxed embedding preserves only the ancestor descendant relationships of the gene tree, rather then the least common ancestor relationships. An example for such a relaxed embedding is depicted in Figure 1.1 where the LCA of a and b in G is r but in R_2 it is y.

Following the objectives of a reconciled tree under the basic GD model, we define (i) a *node reconciled tree* to be an explanation tree in $\mathsf{Exp}_{G,S}$ with the minimum number of nodes and (ii) a *dup-loss reconciled tree* to be an explanation tree with the minimum reconciliation cost in the number of duplication and losses. As we show node and dup-loss reconciled tree are not necessarily identical and unique. Given a gene and its species tree, the *node reconciliation problem* is to find a node reconciled tree and the *dup-loss reconciliation problem* is to find a dup-loss reconciled tree. We show that both problems are solvable in polynomial time.

Outline

This work is organized as follows: In Chapter 2 we briefly sketch the main idea underlying our work. This chapter will outline a rough picture of what we are trying to achieve. The technical aspects of our work are discussed from Chapter 4 to Chapter 7 after introducing necessary definitions and notation in Chapter 3.

2 OVERVIEW OF THE MAIN IDEA

We present an overview of our solution for the node reconciled tree problem that is divided into two subsections. In Section 2 we show that the problem can be divided into independent subproblems whose solutions can be combined into a solution to the GD problem. To solve the subproblems we apply a dynamic programming approach that is outlined in Section 2. The solution for the dup-loss problem is similar and is given in Chapter 7.

First, we introduce some necessary notation. Let G be a gene tree, S its species tree, and R a node reconciled tree for G and S. We introduce three functions that relate those trees pair-wise. Emb denotes a relaxed embedding function that embeds the gene tree into the reconciled tree. The LCA function maps every gene from the given gene tree G to the species with maximal depth (most recent species) in the species tree S, that is able to contain the gene. For example in Figure 1.1 the root r of the gene tree G can not be contained in species Y, but can be contained in the root of the species tree. We say that *a gene g is in species s* if $s = \mathsf{LCA}(g)$. The Dup function maps every gene from a given reconciled tree for G and S to the species in S where it duplicates or speciates. As an example in Figure 1.1 gene y_3 in the reconciled tree speciates in species Y in the reconciled tree.

Divided and Combine

We are dividing the problem into independent subproblems using the LCA-theorem that states: $\mathsf{LCA}(g) = \mathsf{Dup}(\mathsf{Emb}(g))$ for every gene g in the gene tree (see Theorem 4.3).

Thus, the species of a gene in the gene tree G and its embedding in the reconciled tree R are identical. This allows to partition the edges of the gene and species tree into independent subproblems of the node reconciled tree problem, and the edges of the reconciled tree into solutions to these subproblems.

Consider a partition of the edges in the gene tree G into star trees (parent-child edges). Each star tree rooted at node g defines the edges of a subtree in the species tree, called the *environment of g in the species tree S*. This subtree is defined to be rooted at the node $\mathsf{LCA}(g)$ where the subtrees rooted at $\mathsf{LCA}(c)$ for every child c of g are removed. Similarly, we define the *environment of g in the reconciled tree R*.

As a result we partition the original problem instance, the gene and species tree, into subproblems that are all pairs consisting of a start tree in G and its environment in S.

Our claim is that the solution to the subproblem consisting of a star tree rooted at g and its environment in S is the environment of g in R. A cut-and-paste argument verifies this claim. Suppose the environment of g in R is a not a solution, then there exists an explanation tree that solves the given subproblem with fewer nodes. Replacing the environment of g in R with this explanation tree results in an explanation tree for the original problem that has fewer nodes then R.

Hence, the solutions to the subproblems can be combined to a solution to the original problem.

Solving the core problem through dynamic programming

Here we outline a dynamic programming approach for solving the *core problem* that is the node reconciliation problem constrained to the instances where the given gene tree is a star tree. To show that solutions to the core problem exhibit optimal substructure we prove that every solution contains node-reconciled trees of a particular form, called 'normal form'. The normal form allows us to describe the size of a node reconciled tree recursively that can be then computed using dynamic programming.

A node reconciled tree R is in *normal form* if it satisfies two properties. The first property is satisfied if for every subtree R_d in R that is rooted at a duplication node d the following holds: all duplications in R_d without d are located in the leftmost subtree rooted at a copy of d.

To describe the second property, suppose that a node reconciled tree R satisfies the first property. Consider a duplication node d in this tree and its copies $d_1, d_2, \ldots, d_r$ ordered from left to right and let C be the set of all children in the gene star-tree that are embedded into the subtree rooted at d. Each subtree rooted at $d_1, \ldots, d_r$ exists because it contains an embedding from nodes in C. Thus the subtrees partition the set C into r non-empty sets, $C_1, \ldots, C_r$, based on the nodes that are embedded into each subtree. Recall that R satisfies the first property of the normal form. Thus the subtrees rooted at $d_2, \ldots, d_r$, called *non-duplication subtree*, do not contain a duplication. It follows that the species of the genes in a non-duplication subtree form an *anti-chain* in S (no two elements are on a same path). Now the second property of the normal form requires that the elements in the anti-chains in S for each non-duplication subtree are ordered by $\leq$. We define $S_i \leq S_j$ if for any $i \in S_i$ and $j \in S_j$ that are on a same path, i has smaller or equal depth then j in S. We refer to the genes in C that form the ordered anti-chains as *layers*. An example for layers is depicted in Figure 2.1.

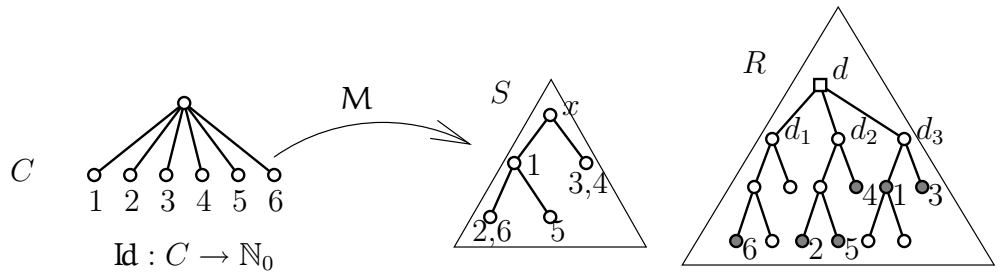

Figure 2.1 The problem instance is simplified to a star gene tree where C is the leaf set. Without loss of generality, a total ordering (Id) in C is assumed. The mapping function M is essentially the LCA mapping function. Layers in C are $\{1, 3\}$, $\{2, 5, 4\}$ and $\{6\}$.

The normal form of a node reconciled tree allows us to describe its number of nodes recursively with respect to the nodes from C that are embedded into the node reconciled tree. Suppose we know that the genes in C are embedded into the subtree R_d of a node reconciled tree in normal form rooted at a duplication node d with copies $d_1, \ldots, d_r$. We refer to the layers of the non-duplication subtrees as $C_2, \ldots C_r$. Then $Size(R_d, C)$ is defined to be the number of nodes in R_d if we embed

C into it. We can write $Size(R_d, C)$ as follows

$$\text{SIZE}(R_d, C) = \text{SIZE}(R_{d_1}, C_1) + (r-1)\text{SIZE}(S_{d'}) + 1$$

where $C_1 = C - \bigcup_{i=2}^{r} C_i$ and d' is the species $Dup(d)$ in S. $\text{SIZE}(S_{d'})$ is the size of each non-duplication subtree, since they do not contain any duplication. We have $(r-1)$ such subtrees. A one is added for the root of R_d and $\text{SIZE}(R_{d_1}, C_1)$ accounts for the size of the remaining subtree rooted R_{d_1}. The layer structure is used to optimize the number of copies, r. A precise formulation of the optimization goes beyond what can be done in this overview. However what was explained here should allow a better understanding of the precise recurrence given in Equation (6.3).

3 NOTATIONS AND DEFINITIONS

This chapter describes fundamental notations used to define the problem formally. The framework follows the principles of the gene duplication problem formalized by Page [17], and includes extensions to cope with the concept of soft multifurcation. Some of their important properties, to be used in the later theorems, are also investigated.

Basic Notations

The following notations are adopted from Cormen *et al.* [5], Semple *et al.* [20], and Bryant [2]. Additional definitions of graphs and trees can be found in appendix A.

$\mathsf{V}(G)$	The vertex (node) set of a tree G.
$\mathsf{E}(G)$	The edge set of a tree G.
$\mathsf{Root}(T)$	The root node of a tree T.
$\mathsf{Le}(T)$	The leaf set of a tree T.
$\mathsf{In}(T)$	The set of internal nodes of a tree T.
$u \leq_T v$	There is a path from u to v in tree T, where u, $v \in \mathsf{V}(T)$.
$u <_T v$	There exists a path from u to v in tree T where u, $v \in \mathsf{V}(T)$ and $u \neq v$.
T_v	The complete subtree T rooted at a node $v \in \mathsf{V}(T)$.
$\mathsf{Depth}_T(v)$	The depth of a node v in a tree T.
$\mathsf{Pa}_T(v)$	The parent of a node v in a tree T.
$\mathsf{Ch}_T(v)$	The set of children of a node v in a tree T.
$\mathsf{Pa}^*_T(v)$	The set of all ancestors of a node v in a tree T.
$\mathsf{Pa}^+_T(v)$	The set $\mathsf{Pa}^*_T(v) - \{v\}$.
$\mathsf{Ch}^*_T(v)$	The set of all descendants of a node v in a tree T.
$\mathsf{Ch}^+_T(v)$	The set $\mathsf{Ch}^*_T(v) - \{v\}$.

In the following discussion, let T, T', G and S be trees. Specifically, G is a gene tree, and S is a species tree. For better readability, when there is no ambiguity, subscripts of notations in future discussions will be omitted. However, while defining a notation, all necessary subscripts will be presented clearly.

Definition 3.1 (leaf association). A function $\mathsf{A} : \mathsf{Le}(G) \to \mathsf{V}(S)$ is a *leaf association* from G to S.

In the following discussion, we assume that the function A is a leaf association from G to S.

Definition 3.2 (LCA). The *least common ancestor* or *LCA* of a non-empty subset X of $\mathsf{V}(T)$, denoted by $\mathsf{LCA}_T(X)$, is a node $u \in \bigcap_{v \in X} \mathsf{Pa}^*(v)$ such that for any other $u' \in \bigcap_{v \in X} \mathsf{Pa}^*(v)$, $\mathsf{Depth}(u') \leq \mathsf{Depth}(u)$.

Note that in a tree T, the LCA of a non-empty subset of $V(T)$ always exists and is unique. Hence LCA_T is essentially a function that maps from $\wp(\mathsf{V}(T)) - \emptyset$ to $\mathsf{V}(T)$.

Definition 3.3 (LCA mapping). The *LCA mapping* function from G to S with a leaf association A is a function $\mathsf{LCA}_\mathsf{A} : \mathsf{V}(G) \to \mathsf{V}(S)$ such that

$$\mathsf{LCA}_\mathsf{A}(u) = \begin{cases} \mathsf{A}(u), & \text{if } u \in \mathsf{Le}(G); \\ \mathsf{LCA}_S(X), & \text{where } X = \mathsf{LCA}_\mathsf{A}(\mathsf{Ch}_G(u)) \text{ and } u \in \mathsf{In}(G). \end{cases}$$

To compute the LCA mapping, reference is made to the algorithms designed by Zhang [22], Eulenstein [6], Bender [1], or Eddy and Zmasek [23]. Gusfield [12] discussed the LCA query algorithm presented by Harel and Tarjan [13] or Schieber and Vishkin [19], which provides the LCA mapping in a gene duplication problem analysis by Zhang.

Definition 3.4 (subtrees). A tree T' is called a *subtree* of T if any pair of nodes $\{u, v\}$ in $\mathsf{V}(T')$, then it holds that $\mathsf{LCA}_T(\{u, v\}) = \mathsf{LCA}_{T'}(\{u, v\})$.

In the above definition, if T' is a subtree of T, it implies that $\mathsf{V}(T') \subseteq \mathsf{V}(T)$, and all ancestor-descendant relations in $\mathsf{V}(T')$ can be found in $\mathsf{V}(T)$. However, the similar result does not apply to edge sets, i.e., $\mathsf{E}(T')$ is not necessarily a subset of $\mathsf{E}(T)$. An intuitive way to obtain a subtree is first to remove unwanted leaf nodes and their adjacent edges repeatedly, then remove internal nodes with only a single child, and finally add an edge to replace the two edges removed along the removed internal node.

Essentially, a node u in $\mathsf{V}(T)$ is missing in $\mathsf{V}(T')$ because there are less than two children of u in $\mathsf{V}(T')$. Also note that by definition, the complete subtree T_v is a subtree of T, for any $v \in \mathsf{V}(T)$.

Definition 3.5 (restriction). Let X be a non-empty subset of $\mathsf{V}(T)$ such that for any x, $y \in X$, $x \not< y$. A tree T' is the *restriction of T to X*, denoted by $T' = T_{|X}$, if T' is a subtree of T and $\mathsf{Le}(T') = \max_T(X)$.

Generally, a restriction of a tree is defined as a subset of its leaf set. Note that by the definition of subtrees, $\mathsf{Root}(T|_X) = \mathsf{LCA}(X)$.

Definition 3.6 (refinement). A tree T is called a *refinement* of T', denoted by $T' \leq T$, if $\mathsf{Le}(T') = \mathsf{Le}(T)$ and $u \leq_{T'} v \Leftrightarrow u \leq_T v$ for any u, $v \in \mathsf{V}(T')$.

Note that if T is a refinement of T', then $\mathsf{V}(T') \subseteq \mathsf{V}(T)$. If T is a binary tree, then there is no other refinement of T except itself; conversely, any tree T is a refinement of a star-tree T' where $\mathsf{Root}(T) = \mathsf{Root}(T')$ and $\mathsf{Le}(T) = \mathsf{Le}(T')$.

Duplication

One important aspect of the gene duplication model, as its name suggests, is duplicating genes. Since genes are hosted in species, such duplication events can be considered to happen at some point during species evolution history. In this section a simple relation is defined to express what a resulting gene phylogeny should be if some gene duplication events happen.

Definition 3.7 (duplication tree). A tree D is called a *duplication tree* of S if there is a surjective (onto) function $\mathsf{Dup} : \mathsf{V}(D) \to \mathsf{V}(S)$, called a *duplication function*, satisfying the following properties:

- $\mathsf{Dup}(\mathsf{Le}(D)) = \mathsf{Le}(S)$.
- If $u \in \mathsf{In}(D)$, then exactly one of these two cases holds for u:
  1. $\mathsf{Dup}(\mathsf{Ch}_D(u)) = \{\mathsf{Dup}(u)\}$, where u is called a *duplication node* (or *d-node*) and $v \in \mathsf{Ch}_D(u)$ a *copy* of u; or
  2. $\mathsf{Dup}(\mathsf{Ch}_D(u)) = \mathsf{Ch}_S(\mathsf{Dup}(u))$, where u is called a *speciation node* (or *s-node*).

Embedding

Since the gene duplication model also involves fitting a gene phylogeny into a species phylogeny, the concept of fitting, or embedding as it is called here, needs to be formalized.

Definition 3.8 (embedding tree). A tree E is called an *embedding tree* of G if there is a function $\mathsf{Emb} : \mathsf{V}(G) \rightarrow \mathsf{V}(E)$, called an *embedding function*, satisfying at least one of the following properties:

1. Emb is an isomorphism from G to E', where E' is the restriction of E to $\mathsf{Emb}(\mathsf{V}(G))$. In this case, Emb is called a *strict embedding*.
2. Emb is an isomorphism from G to E', where the restriction of E to $\mathsf{Emb}(\mathsf{V}(G))$ is a refinement of E'. In this case, Emb is called a *relaxed embedding*.

In the above definition, although Emb is bijective (one-to-one correspondence) from G to E', it is only guaranteed to be injective (one-to-one) from G to E in both strict or relaxed embedding.

For a given pair of trees G and E such that E is an embedding tree of G, the set of all valid strict embedding functions from G to E is always a subset of all relaxed embedding functions from G to E. As mentioned in the definition of a refinement tree, G has to be multifurcated in order to have a refinement tree that is different from itself. Hence it is only possible to have a relaxed embedding function from G to E if G is multifurcated.

Similar to categorizing the nodes of a duplication tree, nodes of an embedding tree are also categorized. These properties will be referred to in later discussions.

Definition 3.9 (categorize embedded nodes). Let E be an embedding tree of G under an embedding function Emb. For a node v in $\mathsf{V}(E)$, it is labeled according to the following rules.

- If v is in $\mathsf{Emb}(\mathsf{V}(G))$, then v is *embedded*, otherwise, v is *un-embedded*.
- A node v is *embedding free* if any $u \in \mathsf{V}(E_v)$ is un-embedded.

Reconciled Trees

As defined by Page [17], the reconciliation between a gene phylogeny and a species phylogeny must fulfill both duplication and embedding requirements, which were just formally defined. Based on the principle of parsimony, which is often called Occam's Razor, a certain criterion is applied in order to single out one representative solution among potentially satisfying ones. These optimization conditions are necessary to define the so-called reconciled tree.

Definition 3.10 (explanation tree). Let G, S trees with a leaf association A from G to S. A tree R is called an *explanation tree* from G to S with the leaf association A under Emb and Dup if both of the following hold:

- The tree R is a duplication tree of S under a duplication function Dup, and R is an embedding tree of G under an embedding function Emb.
- For all node g in $\mathsf{Le}(G)$, $\mathsf{A}(g) = \mathsf{Dup} \circ \mathsf{Emb}(g)$.

The collection of all explanation trees from G to S with a leaf association A is denoted by $\mathsf{Exp}_{\mathsf{A}}(G, S)$.

Although an explanation tree provides the means of reconciliation, there is no optimization involved. Based on the definition by Page [17], a reconciled tree is optimized by the number of nodes in the tree, while in other works by Goodman *et al.* [10] or Guigó *et al.* [11], it is implicitly optimized by the cost of duplications and losses.

The cost of duplications and losses is defined next.

Definition 3.11 (duplication cost). Let R be an explanation tree from G to S. The duplication cost of R, denoted by $C_{\mathsf{d}}(R)$, is defined as $\sum(\mathsf{Deg}(d) - 1)$, where d is a d-node in R.

By the concept of soft multifurcation, a d-node in R is allowed to have at least two copies. Hence it is reasonable to weight each d-node by the number of copies it has. In order to be consistent with previous works, the duplication cost of each d-node is its degree subtracted by one. This is because a binary tree with k leaves has $(k-1)$ internal nodes.

Definition 3.12 (loss). Let R be an explanation tree from G to S. A node v in $\mathsf{V}(R)$ is a *loss* if v is embedding free but $\mathsf{Pa}(v)$ is not.

The *total losses* of R, denoted by $C_{\mathsf{l}}(R)$, is defined as $\big|\{v \in \mathsf{V}(R) \mid v \text{ is a loss in } R\}\big|$.

Definition 3.13 (duplication and loss). The *duplication and loss cost* of R is the sum of duplication cost and total losses of R, denoted by $C_{\mathsf{d+l}}(R)$.

Note that in the above notations of duplication cost and losses, the existence of a duplication function and an embedding function is implied since R is a explanation tree.

Definition 3.14 (node reconciled tree). A *node reconciled tree* from G to S with a leaf association A is a tree $R \in \mathsf{Exp}_{\mathsf{A}}(G, S)$ such that any tree $R' \in \mathsf{Exp}_{\mathsf{A}}(G, S)$, $|R'| \geq |R|$. The collection of all node reconciled trees from G to S with a leaf association A is denoted by $\mathsf{Rec}^1_{\mathsf{A}}(G, S)$.

Definition 3.15 (dup-loss reconciled tree). A *dup-loss reconciled tree* from G to S with a leaf association A is a tree $R \in \mathsf{Exp}_{\mathsf{A}}(G, S)$ such that for any tree $R' \in \mathsf{Exp}_{\mathsf{A}}(G, S)$, it holds $C_{\mathsf{d+l}}(R') \geq C_{\mathsf{d+l}}(R)$. The set of all dup-loss reconciled trees from G to S with a leaf association A is denoted by $\mathsf{Rec}^2_{\mathsf{A}}(G, S)$.

Without specifying the condition, $\mathsf{Rec}_{\mathsf{A}}(G, S)$ denotes the collection of all reconciled trees from G to S with a leaf association A. In other words, $\mathsf{Rec}_{\mathsf{A}}(G, S) = \mathsf{Rec}^1_{\mathsf{A}}(G, S) \cup \mathsf{Rec}^2_{\mathsf{A}}(G, S)$.

Problem Definition

Now the problem to be solved in the later chapters can be defined formally. The problem, called the gene duplication problem with soft multifurcation, will be referred to as *soft GDP*.

Given: a gene tree G, a species tree S and a leaf association A from G to S.

Find: a reconciled tree R from G to S with the leaf association A.

By the definition of reconciled trees, if $R \in \mathsf{Rec}_{\mathsf{A}}(G, S)$ is under an embedding function Emb, then there is no d-nodes in R_v where $v \in \mathsf{Emb}(\mathsf{Le}(G))$ and v can be an internal node in R. Generally in practice, it is expected that the leaf set of S is a subset of $\mathsf{A}(\mathsf{Le}(G))$ since this means that all taxa in the species tree is supported by (at least) one gene taxon. Otherwise, S is replaced by $S_{|\,\mathsf{A}(\mathsf{Le}(G))}$. This also implies $\mathsf{Root}(S) = \mathsf{LCA}_S(\mathsf{A}(\mathsf{Le}(G)))$. However, for the correctness of this analysis, the expected condition and its implication are not necessary.

Basic Observations

Before starting to solve the problem, based on the definitions in this chapter, some properties are derived to help build theorems in later discussions. Most properties mentioned here are rather trivial; hence the proofs of correctness are provided in Appendix A.

Lemma 3.16 (embedding representatives). *Let g be an internal node of G. There exist two children c_1, c_2 of g such that $\mathsf{Emb}(g) = \mathsf{LCA}(\{\mathsf{Emb}(c_1), \mathsf{Emb}(c_2)\})$.*

Lemma 3.17 (naive explanation tree). *There exists an explanation tree from G to S with a leaf association A.*

Note that the above lemma implies the existence of a reconciled tree.

Lemma 3.18 (duplication and embedding path implication). *Let u, v be nodes in $\mathsf{V}(G)$, $u'' = \mathsf{Dup} \circ \mathsf{Emb}(u)$ and $v'' = \mathsf{Dup} \circ \mathsf{Emb}(v)$. If $u \leq v$ then $u'' \leq v''$.*

Corollary 3.19 (LCA mapping lowerbound). *For any node u in $\mathsf{V}(G)$, $\mathsf{Dup} \circ \mathsf{Emb}(u) \leq \mathsf{LCA}(u)$.*

4 STRUCTURAL PROPERTIES OF RECONCILED TREES

Both an embedding function and a duplication function need to be found to determine a reconciled tree. In this chapter, structural properties of a reconciled tree related to these two functions are discussed. Throughout this chapter, let G and S be trees with a leaf association A, and R be a reconciled tree from G to S under Emb and Dup, where Emb is an embedding function from G to R and Dup is a duplication function from R to S. For convenience, unless stated explicitly, if v is a node in $\mathsf{V}(G)$, then v' denotes $\mathsf{Emb}(v)$; for any $v' \in \mathsf{V}(R)$, v'' denotes $\mathsf{Dup}(v')$.

LCA Theorem

The LCA theorem is the first major theorem about structural properties in a reconciled tree. This theorem helps simplify the problem and identify the solution.

Definition 4.1 (LCA equality). For any $g \in \mathsf{V}(G)$, g satisfies *the LCA equality* if and only if $g'' = \mathsf{LCA}(g)$.

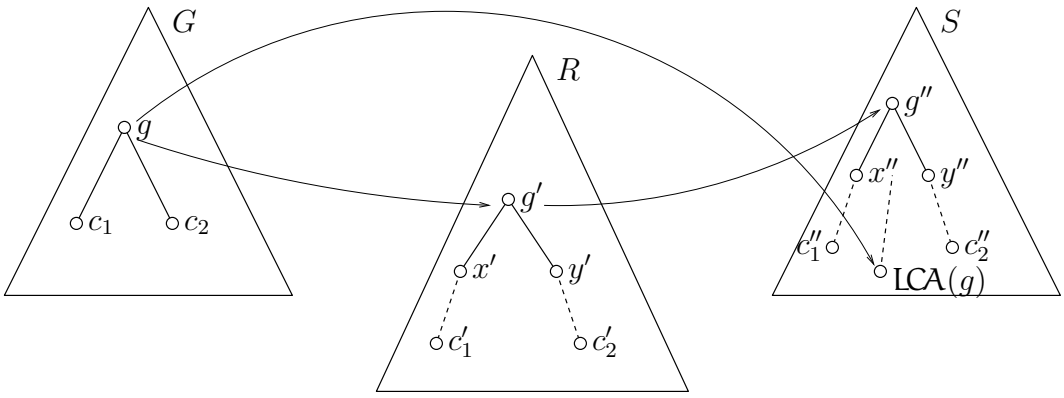

Figure 4.1 The scenario that g' is an s-node in R, and $g'' < \mathsf{LCA}(g)$ in S. Note that without further assumptions, neither $x'' \leq \mathsf{LCA}(g)$ nor $y'' \leq \mathsf{LCA}(g)$ can be presumed.

Lemma 4.2 (d-node LCA inequality). *For any $g \in \mathsf{In}(G)$, if $g'' < \mathsf{LCA}(g)$, then g' is a d-node in R. Furthermore, all copies of g' are un-embedded.*

Proof. (by contradiction, $g'' = \mathsf{LCA}(g)$ if g' is an s-node.)

Let $g \in \mathsf{In}(G)$, $g' = \mathsf{Emb}(g)$. For the purpose of contradiction, assume $g'' < \mathsf{LCA}(g)$ and g' is an s-node in R. Independently, for all $v \in \mathsf{Ch}^+(g)$, it is assumed that $v'' = \mathsf{LCA}(v)$. Since for any leaf

node $v \in \mathsf{Le}(G)$, it is true that $v'' = \mathsf{LCA}(v)$ by definitions. Consequently there exists a node g that satisfies the latter assumption.

By Lemma 3.16, there exists two distinct children c_1, c_2 of g such that $g' = \mathsf{LCA}_R(\{c'_1, c'_2\})$. Let x', $y' \in \mathsf{Ch}(g')$ such that x' and y' are on the path from g' to c'_1 and c'_2 respectively. This scenario is illustrated in Figure 4.1.

Since g' is an s-node, x'' and y'' are children of g'' in S, which implies

$$\begin{aligned} g'' &= \mathsf{LCA}_S(\{x'', y''\}) \\ &= \mathsf{LCA}_S(\{c''_1, c''_2\}), \end{aligned}$$

and contradicts the assumption that $g'' < \mathsf{LCA}(g)$.

Now it is known that g' is a d-node. Let $g'_1, \ldots, g'_k$ be copies of g'. Hence $g''_i < \mathsf{LCA}(g)$ because $g'' = g''_i$ for $i \in \{1, \ldots, k\}$. Recall that for all $v \in \mathsf{Ch}^+(g)$, $v'' = \mathsf{LCA}(v)$ and $\mathsf{LCA}(g) \leq \mathsf{LCA}(v)$. Hence $g''_i \neq \mathsf{LCA}(v)$ which implies all copies of g' are un-embedded if $g'' < \mathsf{LCA}(g)$. □

Observe that in the above proof, R is not necessary a reconciled tree, and the LCA inequality may hold in an explanation tree where $\mathsf{Emb}(g)$ is a d-node. It will be shown that only the equality holds in reconciled trees.

Theorem 4.3 (LCA equality). *For any node $g \in \mathsf{V}(G)$, $\mathsf{LCA}(g) = g''$.*

Proof. (by contradiction, R is not optimal if $g'' < \mathsf{LCA}(g)$.)

A similar argument in the proof of Lemma 4.2 is followed. Let $g \in \mathsf{In}(G)$. Assume $g'' < \mathsf{LCA}(g)$, and $v'' = \mathsf{LCA}(v)$ for all $v \in \mathsf{Ch}^+(g)$. Note that g' can not be an s-node in R by Lemma 4.2.

Let $s = \mathsf{LCA}(g)$ and g'_1, g'_2, $\ldots g'_k$ be the copies of g' in R, for some integer $k \geq 2$. Since g' is a d-node, $g'' = g''_i$ for $1 \leq i \leq k$. According to this implication, $g'' < s$, and there exists one child x'' of g'' such that $g'' < x'' \leq s$. Figure 4.3 illustrates the scenario. Recall that by the assumptions, all proper descendants of g map to the descendants of s, under both LCA and $\mathsf{Dup} \circ \mathsf{Emb}$. There exists a reconciled tree such that all copies of g' are s-nodes.

There is a reconciled tree such that $g'_1, \ldots, g'_k$ are s-nodes:

> Suppose that g'_j is a d-node for some $j \in \{1, \ldots, k\}$. Then transform R into R' by moving all children of g'_j into children of g' and deleting g'_j from $\mathsf{V}(R')$. This transformation is illustrated in Figure 4.2. The resulting R' is still an embedding tree since g'_j is un-embedded.
>
> As a result, R can not be a node reconciled tree since $|R'| = |R| - 1$. Hence if R is a node reconciled tree, all copies of a d-node are s-nodes.
>
> If R is a dup-loss reconciled tree, then R' is still a dup-loss reconciled tree. Note that the procedure can be repeated until a dup-loss reconciled tree is reached where all copies of g' are s-nodes.

By the above claim and the definition of duplication functions, there exists a child x'_i of g'_i such that $\mathsf{Dup}(x'_i) = x''$ for $1 \leq i \leq k$ and $x'' \leq s$. Also note that for any $v \in \mathsf{V}(R_{g'_i}) - \mathsf{V}(R_{x'_i})$, v is un-embedded, otherwise $x'' \leq s$ can not be true.

Now R is transformed into an explanation tree R' in the following steps, which are demonstrated in Figure 4.3.

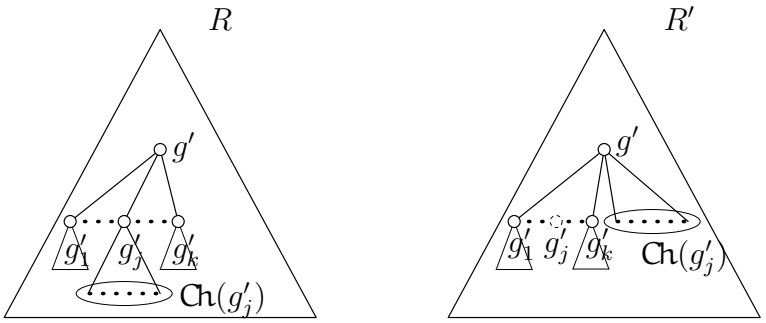

Figure 4.2 A transformation of R into R' by moving all $\mathsf{Ch}(g'_j)$ into $\mathsf{Ch}(g')$ and removing g'_j.

Construction:

The node set $\mathsf{V}(R')$ is obtained from $\mathsf{V}(R)$ by removing all descendants of g'_i except the descendants of x'_i for $2 \le i \le k$. Formally,

$$\begin{aligned} Z &= \bigcup_{2\le i\le k} \mathsf{Ch}^*(g'_i) - \bigcup_{2\le i\le k} \mathsf{Ch}^*(x'_i) \\ \mathsf{V}(R') &= \mathsf{V}(R) - Z. \end{aligned}$$

The edge set $\mathsf{E}(R')$ is also obtained from $\mathsf{E}(R)$ by adding (g', y'_1), for all $y'_1 \in \mathsf{Ch}(g'_1)$ where $y'_1 \neq x'_1$, and (g'_1, x'_i) for all $1 \le i \le k$. For all nodes that are in $\mathsf{V}(R)$ but not in $\mathsf{V}(R')$, their adjacent edges are removed from $\mathsf{E}(R')$ as well. Formally, $\mathsf{E}(R')$ is defined as:

$$\begin{aligned} Y =& \mathsf{Ch}(g'_1) - \{x'_1\} \\ \mathsf{E}(R') =& \{(a,b) \in \mathsf{E}(R) \mid \forall a, b \in \mathsf{V}(R')\} \cup \\ & \{(g'_1, x'_i) \mid \forall i \in \{2, \ldots, k\}\} \cup \\ & \{(g', y) \mid \forall y \in Y\} - \\ & \{(g'_1, y) \mid \forall y \in Y\} \end{aligned}$$

Verification:

The embedding function Emb' from G to R' is obtained from Emb:

$$\mathsf{Emb}'(v) = \begin{cases} g'_1, & \text{if } v = g; \\ \mathsf{Emb}(v), & \text{otherwise.} \end{cases}$$

The mapping of g is changed from g' to g'_1. Since all embedded descendants of g' are transformed as descendants of g'_1, Emb' is a valid embedding function if Emb is valid.

The duplication function Dup' from R' to S is obtained from Dup similar to Emb':

$$\mathsf{Dup}'(v) = \begin{cases} x'', & \text{if } v = g'_1; \\ \mathsf{Dup}(v), & \text{otherwise.} \end{cases}$$

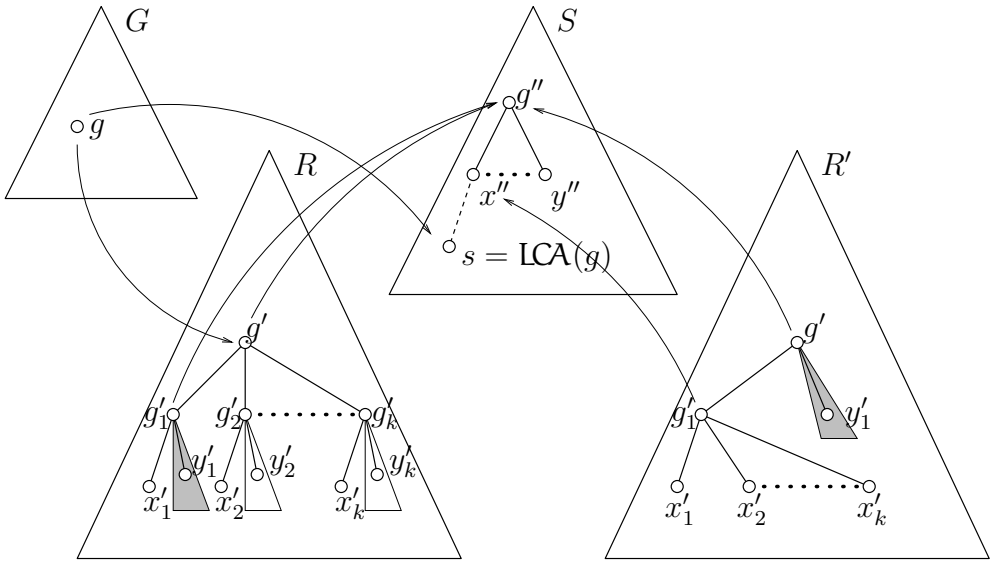

Figure 4.3 The supposed reconciled tree R can be transformed into a better optimized explanation tree R'. As long as $g'' < \mathsf{LCA}(g)$, R can not be a reconciled tree.

In R', g' is no longer a d-node, but g'_1 is a d-node; where $x'' = \mathsf{Dup}'(g'_1)$, with the same number of copies as g' in R.

Since Emb' and Dup' are correct embedding and duplication functions, R' is an explanation tree based on the same embedding type as R. Next R is contradicted as being optimal.

$|R'| < |R|$**:**

Recall that $k \geq 2$ and g is an internal node. There is more than one node in Z, which is the set of nodes removed from $\mathsf{V}(R)$ in $\mathsf{V}(R')$. Hence $|R| - |R'| = |Z| > 0$.

$C_{\mathtt{d+l}}(R') < C_{\mathtt{d+l}}(R)$:

The transformation does not change the duplication cost in R' since g'_1 in R' has the same number of copies as g' in R. However, $y'_2, \ldots, y'_k$ in R are losses which are removed from R'. Thus $C_{\mathtt{d+l}}(R) - C_{\mathtt{d}+eul}(R') = C_{\mathtt{l}}(R) - C_{\mathtt{l}}(R') = (k-1) > 0$.

In conclusion, if R is a reconciled tree from G to S, then for all node $g \in \mathsf{V}(G)$, the LCA equality holds. □

As a result, if the equality does not hold, then R is not a reconciled tree.

Local Reconciled Tree

An explanation tree does not necessarily satisfy the LCA equality and the trees that satisfy it are not necessary reconciled trees. However, an explanation tree that satisfies the LCA equality does have additional structures that it shares with a reconciled tree.

Definition 4.4 (special explanation tree). An explanation tree that satisfies the LCA equality is called a *special explanation tree*. The set of all special explanation trees from G to S with leaf association A is denoted by $\mathsf{Exp}^*_{\mathsf{A}}(G, S)$.

Corollary 4.5 (reconciled tree special). $\mathsf{Rec}_{\mathsf{A}}(G, S) \subseteq \mathsf{Exp}^*_{\mathsf{A}}(G, S)$.

Proof. The statement follows Theorem 4.3 and Definition 4.4. □

Definition 4.6 (explanation cut-out tree). Let tree $T \in \mathsf{Exp}^*_{\mathsf{A}}(G, S)$ under an embedding function Emb. The *cut-out tree of T by a node $g \in \mathsf{V}(G)$*, denoted by $T^{(g)}$, is the complete subtree of T rooted at $\mathsf{Emb}(g)$ with all proper descendant of u removed, for all $u \in \mathsf{Emb}(\mathsf{Ch}(g))$.

For simplicity, the embedding function Emb in the notation of a cut-out tree is omitted. Although an embedding function is not necessarily unique between G and T, a qualified function does exist because of T, and it is assumed to be the same embedding function from the same explanation tree.

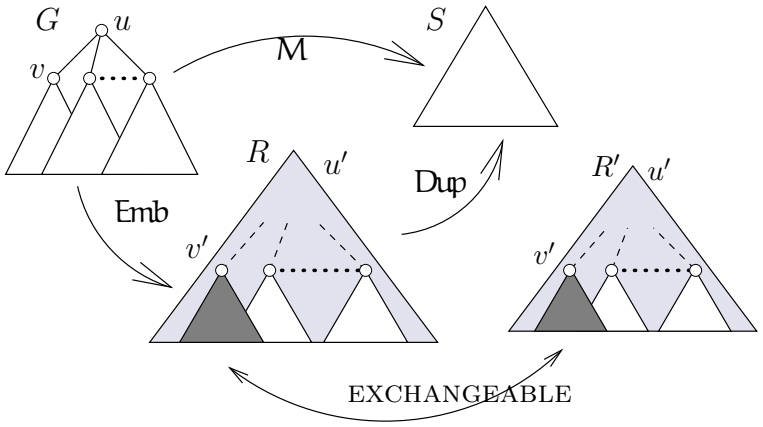

Figure 4.4 In reconciled trees R and R', each parent-children subtree of G corresponds to cut-out trees. Let $u \in \mathsf{In}(G)$, for all $v \in \mathsf{Ch}(u)$, $R_{v'}$ and $R'_{v'}$ are exchangeable. It follows that cut-out trees based on the same internal node of G are exchangeable and optimal. In this figure, M represents an LCA mapping function.

Theorem 4.7 (reconciled tree locally optimal). *Let $T \in \mathsf{Exp}^*_{\mathsf{A}}(G, S)$. For all $g \in \mathsf{In}(G)$ and all $T' \in \mathsf{Exp}^*_{\mathsf{A}}(G, S)$, if $|T^{(g)}| \leq |T'^{(g)}|$ then $T \in \mathsf{Rec}^1_{\mathsf{A}}(G, S)$; if $C_{\mathsf{d+l}}(T^{(g)}) \leq C_{\mathsf{d+l}}(T'^{(g)})$, then $T \in \mathsf{Rec}^2_{\mathsf{A}}(G, S)$.*

Proof. (by contradiction)

Let T be a tree in $\mathsf{Exp}^*_{\mathsf{A}}(G, S)$ such that $|T^{(g)}| \leq |T'^{(g)}|$ for all $g \in \mathsf{In}(G)$ and $T' \in \mathsf{Exp}^*_{\mathsf{A}}(G, S)$. Suppose T is not a node reconciled tree, then there must exist a reconciled tree R such that $|R| < |T|$.

From Corollary 4.5, $R \in \mathsf{Exp}^*_{\mathsf{A}}(G, S)$. The cardinalities of R and T can be derived by summing corresponding cut-out trees. Hence

$$\begin{aligned} |T| &= \sum_{g \in \mathsf{V}(G)} |T^{(g)}| - |G| + 1 \\ &\leq \sum_{g \in \mathsf{V}(G)} |R^{(g)}| - |G| + 1 = |R|, \end{aligned}$$

which is a contradiction.

Similarly, in the case of a dup-loss reconciled tree, we can follow the same argument to reach a contradiction by replacing $R^{(u)}$ with $T^{(u)}$ if $C_{\mathtt{d+l}}(R) < C_{\mathtt{d+l}}(T)$ but $C_{\mathtt{d+l}}(R^{(u)}) > C_{\mathtt{d+l}}(T^{(u)})$. □

Corollary 4.8 (reconciled subtree optimal). *Let R be a reconciled tree from G to S. For any $g \in \mathsf{In}(G)$, $R_{\mathsf{Emb}(g)}$ is a reconciled tree from G_g to $S_{\mathsf{LCA}(g)}$.*

Proof. The statement follows Theorem 4.7. □

The Core Problem

Based on the statements above, the soft GDP can be solved by finding cut-out trees as a local solution and then putting them together as a global solution. The idea of cut-out trees and local reconciled trees is represented in Figure 4.4. Hence the original soft GDP definition is refined to reflect such local solutions under relaxed embedding, and redefined problem is referred to as *the core problem*.

Given: A star-tree G where $C = \mathsf{Le}(G)$, a tree S and a mapping function $\mathsf{M} : \mathsf{V}(G) \to \mathsf{V}(S)$ such that $\mathsf{M}(\mathsf{Root}(G)) = \mathsf{LCA}_S(\mathsf{M}(C))$.

Find: A reconciled tree from G to S with the leaf association M.

According to the given condition, the mapping function M is a leaf association from G to S as well as an LCA mapping function.

One particular benefit of reducing the problem to the core problem is that the definition of an embedding function can be simplified as well. Since G is a star-tree in the core problem, for any distinct u, $v \in C$, Emb is a relaxed embedding function if $\mathsf{Emb}(u) \not\leq \mathsf{Emb}(v)$, and Emb is a strict embedding function if $\mathsf{Emb}(\mathsf{Root}(G)) = \mathsf{LCA}(\{\mathsf{Emb}(u), \mathsf{Emb}(v)\})$.

The notation $\mathsf{Rec}_{\mathsf{M}}(C, S)$ is used to represent the collection of reconciled trees in the core problem, in order to emphasis the leaf set and distinguish from the general soft GDP.

Duplication Criteria

Although the original problem has been successfully reduced to the core problem, the conditions for having d-nodes in a reconciled tree have yet to be determined. Since potentially S might be a reconciled tree, the following equivalent statements are considered, which follow from the definitions of embedding functions.

- There is no d-node in R.

- R and S are isomorphic.
- LCA is an embedding function from G to S.

Hence, if the function LCA can not satisfy the conditions of an embedding function, there has to be a d-node in R. Since each cut-out subtree of a reconciled tree is optimal, the cut-out environment for embedding can also be applied to simplify the discussion.

To correspond to a cut-out subtree, an internal node $g \in \mathsf{In}(G)$ and $C = \mathsf{Ch}(g)$ is considered. Thus $G_{|C}$ forms a basic unit of tree G with respect to $R^{(g)}$. As mentioned above, if LCA violates the rules of embedding functions, then there are two possible cases:

1. LCA is not a strict embedding function;
2. LCA is not a relaxed embedding function.

From these two cases, the following conditions are formalized.

1. There exist u, $v \in \mathsf{LCA}(C)$ such that $\mathsf{LCA}(g) \neq \mathsf{LCA}_S(\{u, v\})$;
2. There exist u, $v \in \mathsf{LCA}(C)$ such that $u \leq v$.

As mentioned in the core problem, the above Condition 2 is sufficient to indicate a violation of the relaxed embedding function. As for the strict embedding, the additional Condition 1 must be taken into account as well since strict embedding functions are defined as special cases of relaxed embedding functions.

Lemma 4.9 (strict embedding uniqueness). *Under strict embedding,* $\mathsf{Rec}^1(G, S) = \mathsf{Rec}^2(G, S)$. *Furthermore,* $|\mathsf{Rec}(G, S)| = 1$.

Proof. (direct proof)

Let $g \in \mathsf{In}(G)$ and $C = \mathsf{Ch}(g)$. Consider the star-tree $G_{|C}$. The corresponding cut-out tree $R^{(g)}$ must be a star-tree ($R_{|\,\mathsf{Emb}(C)}$) as well, since Emb is a strict embedding function.

As a result, only $\mathsf{Emb}(g)$ can be a d-node in $R^{(g)}$ if LCA is not a an embedding function or C violates any duplication criterion. Once $\mathsf{Emb}(g)$ is a d-node, it has exactly $|C|$ copies in order to form an embedding with least cost based on either optimization condition.

Hence the resulting reconciled tree is unique. □

The duplication criterion for a gene duplication problem under strict embedding, introduced here, is consistent with the results proposed by Page [17] and proven by Eulenstein [7] independently. If G is a binary tree, there is only one relaxed embedding function which is the same as the strict embedding function. This implies the uniqueness of the reconciled tree if G is binary.

The following discussions will focus on the cases where G is multifurcated and embedding functions are relaxed embedding.

5 CORE RECONCILED TREES IN NORMAL FORM

In the core problem, the input tree G is a star-tree with leaf set C, and R is in $\mathsf{Rec}_{\mathsf{M}}(C, S)$ under an embedding function Emb and a duplication function Dup. Figures 5.1 and 5.2 illustrate that reconciled trees may not be unique. In this chapter a *normal form* of reconciled trees is introduced and solutions to find reconciled trees in such a normal form are presented later.

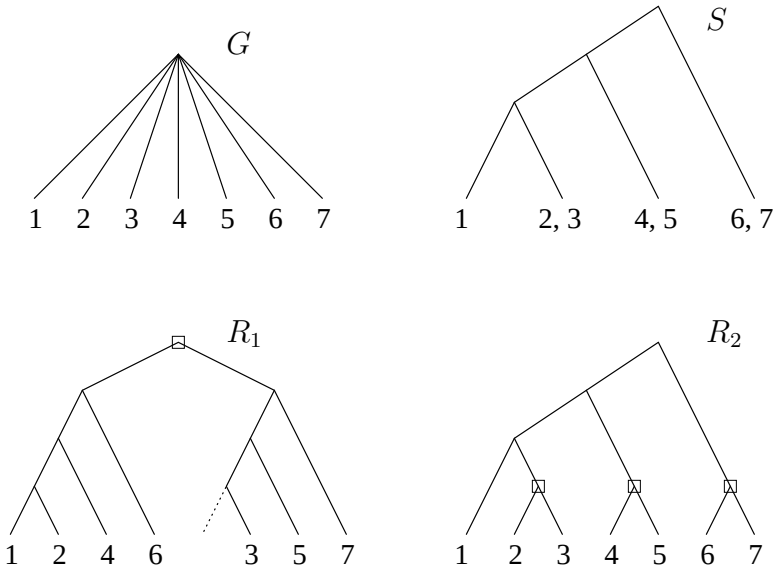

Figure 5.1 Different reconciled trees by different optimization conditions. R_1 has 1 duplication and 1 loss, but $|R_1| = 15$. On the other hand, R_2 has 3 duplications without any losses, and $|R_2| = 13$.

The Normal Form

Definition 5.1 (reconciled trees in normal form). [1] A reconciled tree R from G to S is in *normal form* if there is no d-node in $\mathsf{V}(R)$, or if any d-node d in $\mathsf{V}(R)$, with copies $d_1, \ldots, d_k$, R satisfies the following properties.

1. **The property of normalized duplication:** There exists no d-node in $\mathsf{V}(R_{d_i})$ for $2 \leq i \leq k$.

[1] Although the definition of the normal form is based on reconciled trees in the core problem, the definition can be extended to general reconciled trees by considering cut-out trees. For simplicity, the focus is on the core reconciled trees.

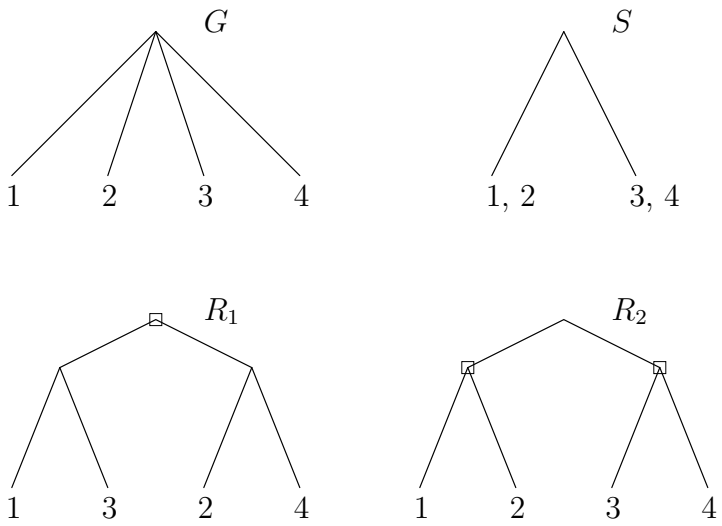

Figure 5.2 Both R_1 and R_2 are reconciled trees from G to S with the same cardinality. However, duplication and loss costs are not the same.

2. **The property of normalized embedding:** For any embedded node u in $\mathsf{V}(R_{d_i})$ where $1 \leq i < k$, there exists a distinct embedded node v in $\mathsf{V}(R_{d_j})$ such that $\mathsf{Dup}(v) \leq \mathsf{Dup}(u)$ for $i < j \leq k$.

Lemma 5.2 (d-node uniqueness). *There exists a reconciled tree R such that for all $s \in \mathsf{V}(S)$, at most one d-node d in R is mapped to s under* Dup.

Proof. (by contradiction)

Suppose u, $v \in \mathsf{V}(R)$ are distinct d-nodes such that $\mathsf{Dup}(u) = \mathsf{Dup}(v)$, $u_1, \ldots, u_k$ are copies of u and $v_1, \ldots, v_r$ are copies of v as shown in Figure 5.3. Without loss of generality, it is assumed that u is not the root of R since there is only one root node.

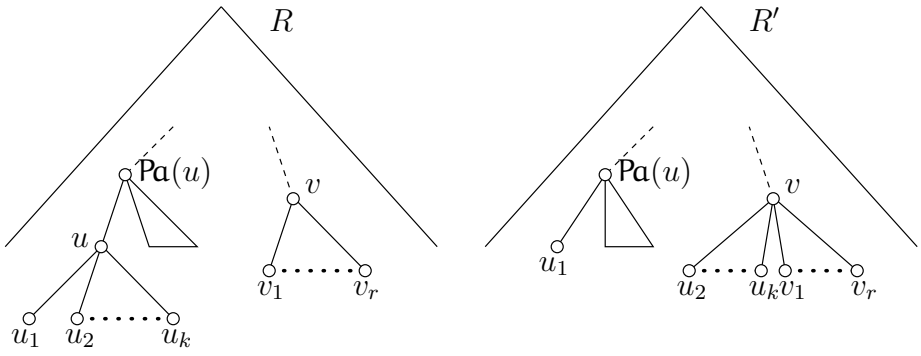

Figure 5.3 The explanation tree R' is transformed from R by connecting $(\mathsf{Pa}(u), u_1)$ and (v, u_i) for all $i \in \{2, \ldots, k\}$ and removing u.

Next a procedure is described to transform R into an explanation tree R' such that $|R'| < |R|$, which contradicts that R is a node reconciled tree.

Transforming R into R':

As depicted in Figure 5.3, v is made the parent of $u_2, \ldots, u_k$. Then u is removed by making an edge $(\mathsf{Pa}(u), u_1)$.

R' is a duplication tree:

The duplication function Dup from R to S is a duplication function from R' to S. It suffices to verify Dup on v and $\mathsf{Pa}(u_1)$ ($\mathsf{Pa}(u)$ in R) only.

Since $\mathsf{Dup}(u_i) = \mathsf{Dup}(v)$ for $2 \leq i \leq k$, v is a d-node in R' as is in R; since $\mathsf{Pa}(u_1)$ in R' is the same as $\mathsf{Pa}(u)$ in R and $\mathsf{Dup}(u_1) = \mathsf{Dup}(u)$, $\mathsf{Ch}(\mathsf{Dup}(\mathsf{Pa}(u_1)))$ is the same as $\mathsf{Ch}(\mathsf{Dup}(\mathsf{Pa}(u)))$. Hence R' is a duplication tree of S.

R' is an embedding tree:

Similarly, the embedding function Emb from G to R can be used as a embedding function from G to R'.

Suppose Emb is not an embedding function from G to R'. There must exist x, $y \in C$ such that $x' = \mathsf{Emb}(x) \leq_{R'} y' = \mathsf{Emb}(y)$. But $x' \not\leq_R y'$ since G is a star-tree. It is only possible if $x' \in \mathsf{Pa}^*(v)$. However, this implies R is not a reconciled tree since v is a d-node.

Contradiction:

With u removed from $\mathsf{V}(R')$, a contradiction is reached because R' is an explanation tree and $|R'| < |R|$. That is, R can not be a node reconciled tree.

In the case where R is a dup-loss reconciled tree, after the transformation, R' is a dup-loss reconciled tree as well. By repeating the transformation procedure whenever possible, a dup-loss reconciled tree is obtained such that there is at most one d-node that maps to s, for $s \in \mathsf{V}(S)$. Since there are only finite nodes mapped to s, the procedure eventually terminates. □

Lemma 5.2 helps to reduce the search space of (node) reconciled trees since an explanation tree can not be a node reconciled tree if the lemma is violated. For simplicity, the search of dup-loss reconciled trees is also limited to the ones that satisfy the lemma.

Since a reconciled tree without any d-node is in formal form by definition, it is assumed there is a d-node in a reconciled tree to exclude the trivial case in later discussion.

An operation, called *switch*, on two nodes x and y in a tree is informally but clearly defined by making $\mathsf{Pa}(x)$ the parent of y and vice versa, where there is no path between x and y and neither is the root node of the tree.

Lemma 5.3 (normalized duplication). *There exists a reconciled tree that satisfies the normalized duplication property.*

Proof. (by construction)

Let d be a d-node in a reconciled tree R with copies $d_1, \ldots, d_k$ where $k \geq 2$. Assume v is a d-node with copies $v_1, \ldots, v_r$ in a subtree R_{d_i} where $2 \leq i \leq k$ and $r \geq 2$. Because d is a d-node, there exist nodes v', $u_1 = \mathsf{Pa}(v')$ and $u_i = \mathsf{Pa}(v)$ such that $\mathsf{Dup}(v') = \mathsf{Dup}(v)$ and $\mathsf{Dup}(u_1) = \mathsf{Dup}(u_i)$.

The tree R in Figure 5.4 is one such example. The existence of the d-node v violates the normalized duplication property.

A transformation of R is shown that results in another reconciled tree R' where v does not violate the normalized duplication property.

Since R is a reconciled tree, $\mathtt{Pa}^*(v) \cap \mathtt{Emb}(C) = \emptyset$ or it is said they are un-embedded from C. Otherwise, v can be replaced by an s-node which implies R is not a reconciled tree. The same statement may or may not hold for the nodes on the path from d to u_1, which will be discussed in two cases.

Case I:

Let $x \in C$ be a node such that $x' = \mathtt{Emb}(x)$ is on the path from d to u_1. The reconciled tree R is transformed into another reconciled tree R' in the following steps where v does not violate the normalized duplication property.

Transforming R into R':

Because R is a duplication tree, there exists a node y' on the path from d to u_i such that $\mathtt{Dup}(x') = \mathtt{Dup}(y')$. Tree R is transformed into R' by switching x' and y'. The transformation is shown in Figure 5.4.

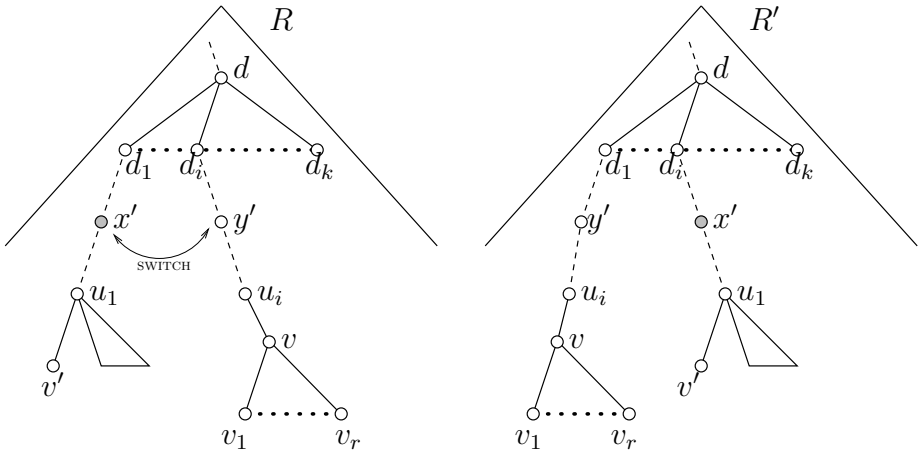

Figure 5.4 The explanation R' is transformed from R by switching x' and y'.

R' is a duplication tree:

The duplication function $\mathtt{Dup}$ from R to S is a duplication function from R' to S because $\mathtt{Dup}(x') = \mathtt{Dup}(y')$.

R' is an embedding tree:

The embedding function $\mathtt{Emb}$ from G to R is an embedding function from G to R' because all nodes in $\mathtt{Pa}^+(x')$ and $\mathtt{Pa}^+(y')$ are un-embedded in both R and R' except $\mathtt{Emb}(\mathtt{Root}(G))$.

Case II:

All nodes on the path from d to u_1 are un-embedded from C. Figure 5.5 explains the following transformation of R into a reconciled tree R'' where v does not violate the normalized duplication property.

Transforming R into R'':

The reconciled tree R is transformed into another reconciled tree in two steps as displayed in Figure 5.5. First R is transformed into a tree T by switching v' and v_1. Next T is transformed into R'' by switching v_1 and v.

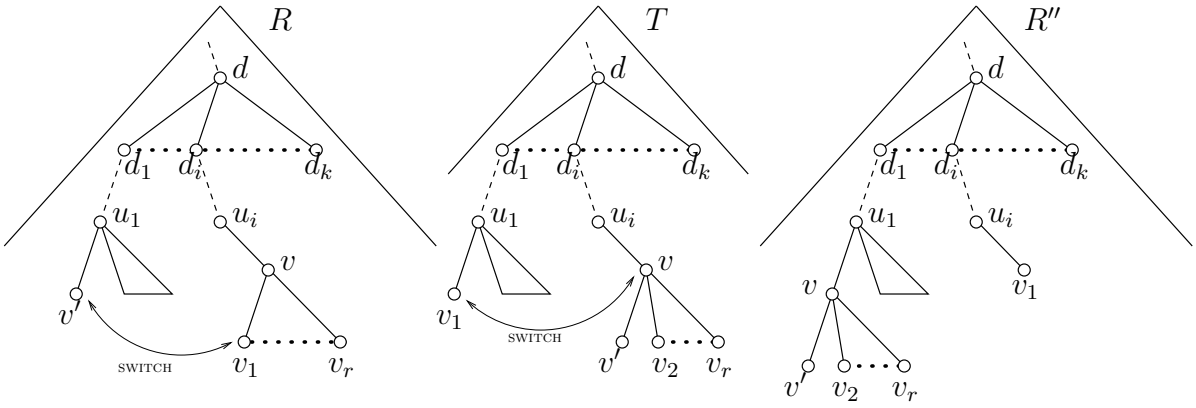

Figure 5.5 A reconciled tree R is transformed into a reconciled tree R''. First R is transformed into T by switching v and v_1. Next T is transformed into R'' by switching v_1 and v.

R'' is a duplication tree:

The duplication function Dup from R to S is a duplication function from R'' to S. It suffices to verify Dup on u_1, u_i and v since they are the only nodes in R'' with different children than in R. Since $\mathsf{Dup}(v') = \mathsf{Dup}(v) = \mathsf{Dup}(v_1)$, these three nodes remain the same d-node or s-node status in R'' as in R. Hence R'' is a duplication tree of S.

R'' is an embedding tree:

The embedding function Emb from G to R is an embedding function from G to R''. It suffices to verify Emb on $\mathsf{Pa}^*(v)$ and $\mathsf{Pa}^*(v_1)$ because of the transformation. Since v and all nodes in $\mathsf{Pa}^*(u_1) \cup \mathsf{Pa}^*(u_i)$ are un-embedded except $\mathsf{Emb}(\mathsf{Root}(G))$, Emb is an embedding function from G to R''.

In the two cases above, $|R| = |R'|$ and $|R| = |R''|$, and the duplication and loss costs remain the same. The transformed explanation trees are reconciled trees. Most importantly, v no longer violates the normalized duplication property in both R' and R'' as in R. Note that due to the transformations, all d-nodes in R_{d_1} (if there is any) are in R'_{d_1} or R''_{d_1} as well.

By applying the procedure of transformation repeatedly, all d-nodes in R_{d_i} can be moved, where $i > 1$, into R_{d_1} in a finite number of steps, since there are only finite d-nodes in the whole reconciled tree, which gives a reconciled tree satisfying the normalized duplication property. □

Theorem 5.4 (reconciled tree in normal form). *There exists a reconciled tree in normal form.*

Proof. (by construction)

By Lemma 5.3, it suffices to transform a reconciled tree with the normalized duplication property into a reconciled tree in normal form.

As illustrated in Figure 5.6, let d be a d-node in R with copies $d_1, \ldots, d_k$, where there are no d-nodes in R_{d_i} for $i > 1$. Assume that there exists two embedded nodes u_i and v_j in R_{d_i} and R_{d_j} respectively such that $\mathsf{Dup}(u_i) <_S \mathsf{Dup}(v_j)$ where $1 \leq i < j \leq k$. The pair $\{u_i, v_j\}$ violates the normalized embedding property. Let $u' = \mathsf{Dup}(u_i)$ and $v' = \mathsf{Dup}(v_j)$.

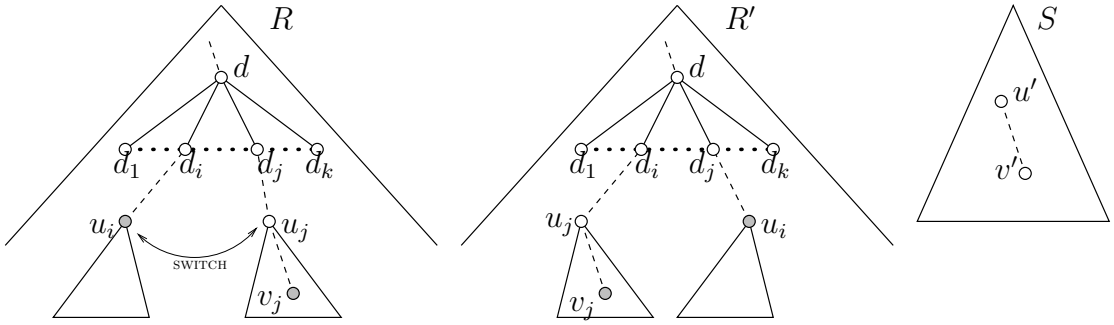

Figure 5.6 The explanation tree R' is transformed from R by switching u_i and u_j.

The reconciled tree R is transformed into R' where the pair $\{u_i, v_j\}$ does not violate the normalized embedding property and the normalized duplication property is preserved.

For the transformation, a node u_j is found in R_{d_j} such that $\mathsf{Dup}(u_j) = u'$ and $u_j < v_j$. The node u_j exists because R is a duplication tree of S,

Transforming R into R':

The reconciled tree R is transformed into R' by switching u_i and u_j as depicted in Figure 5.6.

R' is a duplication tree:

The duplication function Dup from R to S is a duplication function from R' to S since $\mathsf{Dup}(u_i) = \mathsf{Dup}(u_j)$. Both parent nodes of u_i and u_j remain the same d-node or s-node status in R'.

R' is an embedding tree:

The embedding function Emb from G to R is an embedding function from G to R'. In R, u_i and v_j are the only embedded nodes on the path from d to u_i and v_j respectively, which remains the same in R'.

Because $|R| = |R'|$ and the duplication and loss costs are the same, R' is a reconciled tree from G to S. There is no d-nodes in R_{u_i} since u_i is embedded, hence R' is still a reconciled tree with the normalized duplication property. In R', the pair $\{u_i, v_j\}$ does not violate the normalized embedding property in R'.

To argue that R' satisfies the normalized duplication property if R does, it suffices to show that there is no d-nodes in R_{u_i}. Since u_i is an embedded node, any d-node in R_{u_i} implies R is not a reconciled tree.

In addition, the number of pairs violating the normalized embedding property in R' should be less than that number in R. To guarantee that, the copies d_i and d_j has to be picked with one condition such that for any embedded node $w_l \in \mathsf{V}(R_{d_l})$, it holds that $\mathsf{Dup}(u_i) \leq \mathsf{Dup}(w_l) \leq \mathsf{Dup}(v_j)$. Otherwise, either $\{u_i, w_l\}$ or $\{w_l, v_j\}$ becomes the pair to perform the transformation. As long as there exists a pair of violating embedded nodes, a proper pair for transformation to reduce violations can always be found.

Because there can only be finite pairs of nodes that violate normalized embedding, a reconciled tree is eventually obtained that satisfies the normalized embedding property from the original one within a finite number of transformations. Since the normalized duplication property is not affected during the transformations, a reconciled tree in normal form is obtained. □

In conclusion, all copies $d_1, \ldots, d_k$ of a d-node d are essentially *rearranged* such that all R_{d_i} are d-nodes free except R_{d_1}, and R_{d_j} has embedded node with less depth than R_{d_i} if $i < j$ in a reconciled tree.

Optimal Substructure

First a new notation is introduced to specify subsets of C by R and S based on the idea of reconciled trees in normal form.

Definition 5.5 (substructure of C). Let v be a node in $\mathsf{V}(R)$ and u be a node in $\mathsf{V}(S)$. Similar to the idea of complete subtree, $C_v = \{x \in C \mid \mathsf{Emb}(x) \in \mathsf{V}(R_v)\}$ and $C_u = \{x \in C \mid \mathsf{LCA}(x) \in \mathsf{V}(S_u)\}$.

Lemma 5.6 (Reconciled tree optimal substructure). *For any node $v \in \mathsf{V}(R)$, R_v is a reconciled tree from $G_{|C_v}$ to $S_{\mathsf{Dup}(v)}$. In other words, $R_v \in \mathsf{Rec}(C_v, S_{\mathsf{Dup}(v)})$.*

Proof. (direct proof)

Let $G' = G_{|C_v}$ and $S' = S_{\mathsf{Dup}(v)}$. Note that the mapping functions from $\mathsf{V}(G)$ to $\mathsf{V}(S)$ and the one from $\mathsf{V}(G')$ to $\mathsf{V}(S')$ are not necessarily the same, particularly for the root nodes. However, if both instances are represented as the core problem, then the mapping function can be reused directly.

By the definition of duplication functions, it is known R_v is a duplication tree of S' under Dup since R is a duplication tree of S.

It is known that R is an embedding tree of R under Emb. If $C \neq C_v$, let $\mathsf{Emb}(\mathsf{Root}(G') = \mathsf{LCA}(\mathsf{Emb}(C_v)$. Because $\mathsf{Emb}(C_v) \subseteq \mathsf{V}(R_v)$, it follows that $\mathsf{Root}(G') \leq \mathsf{Emb}(\mathsf{Root}(G_{C_v}))$, and Emb is an embedding function from G' to R_v.

Conversely, embedding and duplication functions from G' to R_v and R_v to S' are parts of embedding and duplication functions from G to R and R to S respectively. Hence if R_v is not a reconciled tree from G' to S', by replacing R_v with an optimal one, it contradicts that R is a reconciled tree from G to S. □

The above lemma shows a basic structure of a recursive approach. That is, to find the reconciled tree rooted at v, the reconciled subtrees rooted at $\mathsf{Ch}(v)$ should be found first. Note that this property is always true for R regardless of whether R is in the normal form or not.

Layers and Remains

To further realize the recursion, the connection between C_v and C_u is then established, where $u \in \mathsf{Ch}_R(v)$. Without loss of generality, it is assumed there is a total ordering on C defined by an injective (onto) function $\mathsf{Id} : C \to \mathbb{Z}^+$.

Definition 5.7 (layer and remain). Let X be a subset of C.

- A node $v \in X$ is called a *top* of X if $\mathsf{LCA}(v) \in \min_S(\mathsf{LCA}(X))$ with minimal $\mathsf{Id}(v)$. In other words, there is no other element $u \in X$ such that $\mathsf{LCA}(u) \in \min_S(\mathsf{LCA}(X))$ and $\mathsf{Id}(u) < \mathsf{Id}(v)$. The set of all top elements of X is denoted by $\textsc{Top}(X)$.
- *The i-th remain* of X, denoted by $\textsc{Remain}(X, i)$, for $i \in \mathbb{N}_0$, is defined as

$$\textsc{Remain}(X, i) = \begin{cases} X, & \text{if } i = 0; \\ \textsc{Remain}(X, i-1) - \textsc{Top}(\textsc{Remain}(X, i-1)), & \text{otherwise.} \end{cases}$$

- *The i-th layer* of X, denoted by $\textsc{Layer}(X, i)$, for $i \in \mathbb{Z}^+$, is defined as

$$\textsc{Layer}(X, i) = \textsc{Top}(\textsc{Remain}(X, i-1)) \ .$$

Definition 5.8 (counting layers). Let v be a node in $\mathsf{V}(S)$. The following two values of a node v can be computed in linear time.

- The *local layers*, $n(v) = \big|\{x \in C \mid \mathsf{LCA}(x) = v\}\big|$, is simply counting the number of elements in C that mapped to v.
- The *total layers*, $m(v)$, is defined recursively:

$$m(v) = \begin{cases} n(v), & \text{if } v \text{ is a leaf;} \\ \max\limits_{u \in \mathsf{Ch}(v)} \{m(u)\} + n(v), & \text{otherwise.} \end{cases}$$

The recursive definition of $m(v)$ is equivalent to counting all non-empty layers of C_v. Later these two numbers will be needed to find reconciled trees.

The following lemma shows how layers and remains connect to the optimal substructure of R just discussed.

Lemma 5.9 (layer substructure). *Let $u \in \mathsf{Ch}_S(v)$ and $x \in C$. If there exist $i \in \mathbb{Z}^+$ and $j \in \mathbb{Z}^+$ such that $x \in \textsc{Layer}(C_u, i)$ and $x \in \textsc{Layer}(C_v, j)$, then $j = i + n(v)$.*

Proof. (direct proof)

Let $w = \text{LCA}(x)$ and P the set of notes on the path from v to w (including v and w). To show the equality, the first i layers of C_u mapped to P are compared to the first j layers of C_v mapped to P. Formally, C'_v and C'_u are defined as

$$C'_u = \{y \in \bigcup_{k=1}^{i} \textsc{Layer}(C_u, k) \mid \text{LCA}(y) \in P\}$$
$$C'_v = \{y \in \bigcup_{k=1}^{j} \textsc{Layer}(C_v, k) \mid \text{LCA}(y) \in P\} \ .$$

According to given conditions, $x \in C'_v$, $x \in C'_u$, $|C'_u| = i$ and $|C'_v| = j$. By Definition 5.7, $C'_u \subseteq C'_v$, since any $y \in C'_u$, either $\text{LCA}(y) \leq \text{LCA}(x)$ or $\text{Id}(y) < \text{Id}(x)$. Conversely, the set

$$C''_v = \{y \in C_v \mid \text{LCA}(y) = v\}$$

consists of elements in C'_v but not in C'_u. We know $|C''_v| = n(v)$.

Hence

$$\begin{aligned} C'_v &= C'_u \cup C''_v \\ |C'_v| &= |C'_u| + |C''_v| \\ j &= i + n(v) \ . \end{aligned}$$

□

The above lemma can be extended for remains with a slight modification.

Corollary 5.10 (remain substructure). *Let $u \in \text{Ch}_S(v)$ and $x \in C$. For some $i \in \mathbb{Z}^+$, $x \in \textsc{Remain}(C_u, i)$ iff $x \in \textsc{Remain}(C_v, i + n(v))$.*

Proof. (by contrapositive) It is equivalent to showing $x \notin \textsc{Remain}(C_u, i)$ iff $x \notin \textsc{Remain}(C_v, i + n(v))$ for $x \in C_u$.

Part I: $x \notin \textsc{Remain}(C_u, i) \Rightarrow x \notin \textsc{Remain}(C_v, i + n(v))$

Because $x \in C_u$ and $x \notin \textsc{Remain}(C_u, i)$, there exist $j \in \mathbb{Z}^+$ such that $x \in \textsc{Layer}(C_u, j)$ where $j \leq i$. By Lemma 5.9, $x \in \textsc{Layer}(C_v, j + n(v))$ where $j + n(v) \leq i + n(v)$. Thus x can not be in $\textsc{Remain}(C_v, i + n(v))$.

Part II: $x \notin \textsc{Remain}(C_u, i) \Leftarrow x \notin \textsc{Remain}(C_v, i + n(v))$

Similarly, since $x \in C_u$ and $x \notin \textsc{Remain}(C_v, i + n(v))$, there exist $j \in \mathbb{Z}^+$ such that $x \in \textsc{Layer}(C_v, j)$ where $j \leq i + n(v)$. By Lemma 5.9, $x \in \textsc{Layer}(C_u, j - n(v))$ where $j - n(v) \leq i$. Thus x can not be in $\textsc{Remain}(C_u, i)$.

□

The following lemma further demonstrates how the definitions of remains and layers can help one understand the optimal substructure.

Theorem 5.11 (layer, remain embedding). *Let v be a vertex in R. C_v is either a layer or a remain of $C_{\mathsf{Dup}(v)}$.*

Proof. (by induction over $h = \mathsf{Depth}_R(v)$)

This is shown by induction from the root node of R and then its descendants.

$h = 0$**:**

It is known that $C_{\mathsf{Root}(R)} = C$, which is exactly the same as $C_{\mathsf{Root}(S)}$ and $\mathsf{Root}(S) = \mathsf{Dup}(\mathsf{Root}(R))$. In other words, $C_{\mathsf{Root}(R)} = \textsc{Remain}(C_{\mathsf{Dup}(\mathsf{Root}(R))}, 0)$.

$h \to h + 1$**:**

Suppose the statement holds for all nodes $w \in \mathsf{V}(R)$ with $\mathsf{Depth}_R(w) \leq h$. Let $u \in \mathsf{V}(R)$ such that $\mathsf{Depth}_R(u) = h + 1$. Since u can not be the root node of R, there exists a node $v = \mathsf{Pa}(u)$ such that $\mathsf{Depth}_R(v) = h$. Two cases are discussed separately where v is a s-node and v is a d-node. For convenience, let $v' = \mathsf{Dup}(v)$ and $u' = \mathsf{Dup}(u)$.

Case I: v is a s-node

Because R is a duplication tree of S, $v' = \mathsf{Pa}_S(u')$. By the induction hypothesis, C_v is either a remain for a layer of $C_{v'}$ and Lemma 5.9, or Corollary 5.10 can be applied accordingly.

If $C_v = \textsc{Layer}(C_{v'}, i)$ for some $i \in \mathbb{Z}^+$, then for all $x \in C_u \subseteq C_v$, $x \in \textsc{Layer}(C_{u'}, i - n(v'))$. Conversely, for all $x \in \textsc{Layer}(C_{u'}, i - n(v'))$, $x \in \textsc{Layer}(C_{v'}, i) = C_v$, this implies $x \in C_u$ because v is a d-node. Hence C_u is a layer of $C_{u'}$.

Similarly, if $C_v = \textsc{Remain}(C_{v'}, i)$ for some $i \in \mathbb{N}_0$, then $C_u = \textsc{Remain}(C_{u'}, i - n(v))$.

Case II: v is a d-node

Let $\{v_1, \ldots, v_k\} = \mathsf{Ch}_R(v)$ for some integer $k > 2$. It is known that u is one of them. Say $u = v_j$ where $1 \leq j \leq k$. Note that C_v can not be a layer of $C_{v'}$, otherwise C_v can be embedded into R_{v_1} completely which contradicts R is a reconciled tree. Thus $C_v = \textsc{Remain}(C_{v'}, i)$ for some $i \in \mathbb{N}_0$.

As discussed earlier in Theorem 5.4, R is a reconciled tree in normal form. According to the properties of the normal form, $\textsc{Layer}(C_v, 1)$ can be embedded in R_{v_k}, $\textsc{Layer}(C_v, 2)$ in $R_{v_{k-1}}$, $\ldots$, and $\textsc{Layer}(C_v, k-1)$ in R_{v_2}. The remaining elements $\textsc{Remain}(C_v, k-1)$ has to be embedded in R_{v_1} as the structure of the normal form.

In the above description of embedding, the first layer of C_v is in fact the $(i+1)$-th layer of $C_{v'}$ and so on (till the $(k-1)$-th layer of C_v) since $C_v = \textsc{Remain}(C_{v'}, i)$. Thus $C_{v_j} = \textsc{Layer}(C_v, k - j + 1) = \textsc{Layer}(C_{v'}, i + k - j + 1)$ for $2 \leq j \leq k$ and $C_{v_1} = \textsc{Remain}(C_v, k-1) = \textsc{Remain}(C_{v'}, i + k - 1)$. If $u = v_j$ where $2 \leq j \leq k$, then $C_u = \textsc{Layer}(C_{u'}, i + k - j + 1 - n(v'))$; if $u = v_1$, then $C_u = \textsc{Remain}(C_{u'}, i + k - 1 - n(v'))$ according to Lemma 5.9 or Corollary 5.10.

Therefore the induction hypothesis holds.

In conclusion, C_v is either a layer or a remain of $C_{\mathsf{Dup}(v)}$ for all $v \in \mathsf{V}(R)$. □

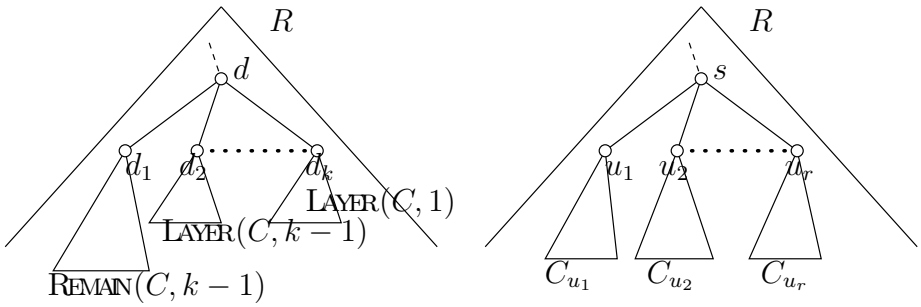

Figure 5.7 Each complete subtree of a reconciled tree R has a layer or remain embedded. In the case that s is an s-node in R, $C_{u_1} \dots C_{u_r}$ form a partition of C_s. In the case that d is a d-node in R, $R_{d_2} \dots R_{d_k}$ are embedded with the firs $k-1$ layers of C_d (denoted as C above).

The lemma and its proof above has outlined the recursive solution as Figure 5.7 demonstrates. As a result, if v is a d-node in R with copies $v_1, \dots, v_k$, layers (excluding the last non-empty remain) will be mapped to the subtree R_{v_i} where $i > 1$. Since there are $k-1$ layers embedded to other copies, the REMAIN$(C_v, k-1)$ can only be embedded to R_{v_1}, and R_{v_1} will be solved recursively in the same fashion.

6 NODE RECONCILED TREE

This chapter provides a solution to finding a node reconciled tree in normal form to the core problem. Structural characterization (LCA equality and the normal form) and optimal substructure (Lemma 5.6, Lemma 5.9 and Corollary 5.10) has been discussed as in a dynamic programming analysis. Next a recursive solution can be designed.

Recall that a set C, a tree S and a function $\mathsf{M} : C \to \mathsf{V}(S)$ have been given.

Recursive Solution

To find a node reconciled tree, a formula is established first to compute its cardinality by assuming it is found. Based on this formula, an optimal solution can be determined by exploring all possibilities of the parameters involved.

Let $c(C, S)$ be the size of a reconciled tree R in $\mathsf{Rec}^1_{\mathsf{M}}(C, S)$ under an embedding function Emb and a duplication Dup, $r = \mathsf{Root}(R)$, and $s = \mathsf{Root}(S)$. Recall that $\mathsf{Dup}(r) = s$. One can start with the base cases where $c(C, S)$ can be determined immediately without relying on subproblems.

If $n(s) = m(s)$, there is no elements in C mapped to $\mathsf{Ch}^+(s)$ at all. This is the base case to determine whether r is a d-node or an s-node. The value $n(s)$ determines two different scenarios.

Case I: $n(s) = m(s) \leq 1$. In this case, r is an s-node and $c(C, S) = |S|$.

Case II: $n(s) = m(s) = k > 1$. In this case, r has to be a d-node with k copies. Hence $c(C, S) = 1 + k \cdot |S|$.

The base case can be rewritten as

$$c(C, S) = \begin{cases} |S|, & \text{if } n(s) = m(s) \leq 1; \\ 1 + k \cdot |S|, & \text{if } n(s) = m(s) = k > 1. \end{cases} \tag{6.1}$$

For the case where $n(s) < m(s)$, subproblems are relied upon to provide the answer according to Lemma 5.6, Corollary 5.10 and Theorem 5.11. The following recursions are illustrated in Figure 5.7. Similar to the base case, r can be an s-node or a d-node, and are discussed separately.

Case I: If r is an s-node, then it implies $n(s) = 0 < m(s)$, and $\{C_u \mid u \in \mathsf{Ch}_R(r)\}$ forms a partition of C. More importantly, $C_u = C_{\mathsf{Dup}(u)}$ and $\mathsf{Dup}(u) \in \mathsf{Ch}_S(s)$ for $u \in \mathsf{Ch}_R(r)$. Thus

$$c(C, S) = 1 + \sum_{u \in \mathsf{Ch}_S(s)} c(C_u, S_u).$$

Case II: If r is a d-node with k copies $r_1, \ldots, r_k$, then $R_{r_2}, \ldots, R_{r_k}$ are isomorphic to S (normalized duplication) and $C_{r_2}, \ldots, C_{r_k}$ are the first $k-1$ layers of C (normalized embedding). Furthermore, r_1 is an s-node in a node reconciled tree (Lemma 5.2), hence R_{r_1} becomes an instance of Case I as shown in Figure 6.1. The recursion can be expressed as

$$\begin{aligned}
c(C,S) &= 1 + (k-1)\cdot|S| + |R_{r_1}| && \text{(normalized duplication)}\\
&= 1 + (k-1)\cdot|S| + c(C_{r_1}, S) && \text{(since } r \text{ is a d-node)}\\
&= 1 + (k-1)\cdot|S| + c(\textsc{Remain}(C, k-1), S) && \text{(normalized embedding)}\\
&= 2 + (k-1)\cdot|S| + \sum_{u\in \mathrm{Ch}(s)} c(\textsc{Remain}(C_u, k-1-n(s)), S_u) && \text{(Corollary 5.10)}
\end{aligned}$$

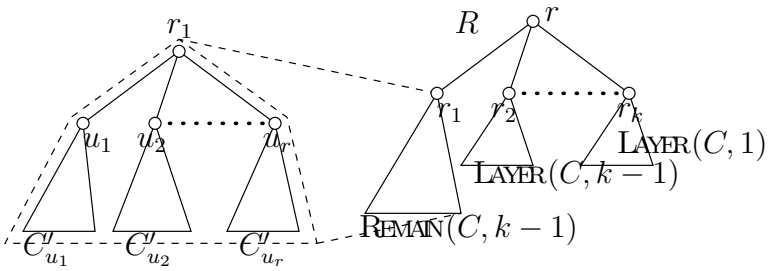

Figure 6.1 In a reconciled tree, the first copy r_1 of a d-node r can be an s-node according to Lemma 5.2.

Note that r is not necessarily an s-node if $n(s) = 0$ and the value of k is not actually known. The intuitive approach is to consider all possible values of k (recall that $k = 1$ represents the case of r being an s-node). Hence the recursive case (chooseing among all reasonable values of k) can be written as

$$c(C,S) = \min_k \Big\{ \mathsf{U}_2(k) + (k-1)\cdot|S| + 1 + \sum_{u\in \mathrm{Ch}(s)} c(\textsc{Remain}(C_u, k-1-n(s)), S_u)\Big\}, \tag{6.2}$$

where U_2 is a modified (shifted) unit step function defined as

$$\mathsf{U}_2(x) = \begin{cases} 1, & \text{if } x \geq 2\\ 0, & \text{otherwise.}\end{cases}$$

A naive lower bound of k is 1 since R is a duplication tree of S. But k can not be smaller than $n(s)$ if $n(s) > 1$ and r is a d-node. In the recursive case ($n(s) < m(s)$), k even has to be greater than $n(s)$. Thus the lower bound of k is $\max\{1, n(s)+1\}$ if $n(s) < m(s)$, but $k = \max\{1, n(s)\}$ if $n(s) = m(s)$.

Recall that R is in normal form, which means if r is a d-node with copies $r_1, \ldots, r_k$, then $C_{r_2}, \ldots, C_{r_k}$ are layers of C. If C_{r_1} is a layer of C, it has to be the last non-empty layer of C, and k reaches its upper bound. Hence the upper bound of k is $m(s)$ since that is the number of non-empty layers in C.

In a simple case , the range of k in (6.2) is $n(s)+1 \leq k \leq m(s)$ (assuming $n(s) \geq 0$). However, in a more general case, as used in a dynamic programming approach without actually initiating a subproblem independently, the range of k has to be choosed with care.

Dynamic Programming

The value k of each node in S should suffice to determine a node reconciled tree. It can be expressed as a function $k : \mathsf{V}(S) \to \mathbb{Z}^+$. To design a dynamic programming algorithm solving the recursion, the connection to subproblems is established according to Corollary 5.10. In other words, the input C and S in (6.2) are always expressed as a remain $\textsc{Remain}(C_v, i)$ and S_v for some node $v \in \mathsf{V}(S)$ and $i \in \mathbb{N}_0$. By replacing $c(\textsc{Remain}(C_v, i), S_v)$ with $\Delta(v, i)$, the recursion can be rewritten as

$$\Delta(v,i) = \begin{cases} \mathsf{U}_2(k_0) + k_0 \cdot |S_v|, & \text{if } n(v) = m(v); \\ \min\limits_{k_1 \leq k \leq k_2} \big\{ \mathsf{U}_2(k) + (k-1) \cdot |S_v| + 1 + \sum\limits_{u \in \mathsf{Ch}_S(v)} \Delta(u, i+k-1-n(v)) \big\}, & \text{otherwise,} \end{cases} \tag{6.3}$$

where constants k_0, k_1 and k_2 are

$$\begin{aligned} k_0 &= \max\{1, n(v) - i\} && \text{(base case; } n(v) = m(v)\text{)} \\ k_1 &= \max\{1, n(v) - i + 1\} && \text{(recursive case lower bound)} \\ k_2 &= \max\{1, m(v) - i\} && \text{(recursive case upper bound)} \end{aligned} \tag{6.4}$$

Recall that the i-th remain of C_v is the subset of C_v with first i layers removed where $i \geq 0$. Hence if $i > m(v) \geq n(v)$, it is handled as an s-node recursively by the definitions of k_0, k_1 and k_2.

The process of the algorithm can be represented by a $|S| \times (m(\mathsf{Root}(S)) + 1)$ table since $v \in \mathsf{V}(S)$ and $0 \leq i \leq m(\mathsf{Root}(S))$. One first needs to know $|S_v|$ for each $v \in \mathsf{V}(S)$. A post-order DFS traversal is a typical choice since $|S_v| = \sum\limits_{u \in \mathsf{Ch}(v)} |S_u|$ if $v \in \mathsf{In}(S)$. All leaf nodes can be initialized immediately since they are all base cases. Each cell stores two values: $\Delta(v, i)$ and $k(v, i)$ as the solution to the subproblems. The table is filled up in a bottom-up fashion. Once the procedure is finished, $\Delta(\mathsf{Root}(S), 0)$ represents $|R|$.

Constructing a Node Reconciled Tree

It suffices to build a node reconciled tree R if $k(v)$ is found for each $v \in \mathsf{V}(S)$. Starting at $\mathsf{Root}(S)$, $k(\mathsf{Root}(S) = k(\mathsf{Root}(S), 0)$ given by the solution. For $v \in \mathsf{In}(S)$, if $k(v) = k(v, i)$, then $k(u) = k(u, i + k(v) - 1 - n(v))$ for all $u \in \mathsf{Ch}(v)$. Essentially $k(v)$ determines a duplication function.

To determine an embedding function, observe that if $k(v) = k(v, i) > 1$, then $d \in \mathsf{V}(R)$ such that $\mathsf{Dup}(d) = v$ is a d-node with $k(v)$ copies, $d_1, \ldots, d_{k(v)}$. The first $k(v) - 1$ layers of $\textsc{Remain}(C_v, i)$ are mapped to the subtree $R_{d_2}, \ldots, R_{d_{k(v)}}$. In case of $n(v) < m(v)$, children of v are advanced recursively as if $k(v) = 1$. Otherwise, it is the $(k(v) + i)$-th layer of C_v mapped to the subtree R_{d_1}.

Note that at the moment, this node reconciled trees are the ones in the core problems. To obtain a reconciled tree in the soft GDP, one relies on Theorem 4.7 and Corollary 4.8 to assemble

local reconciled trees into a global one.

Running Time

The initialization of pre-processing $|S_v|$ takes linear steps. The table in the dynamic programming algorithm takes $O(n^3)$ steps to fill up, where n is the size of input trees. Backtracking the table for $k(v)$ takes $O(n)$ steps only.

Constructing a reconciled tree takes $O(n^2)$ steps after $k(v)$ is known. A duplication function and an embedding function can be determined during the process.

7 DUP-LOSS RECONCILED TREE

This chapter provides a solution to finding a dup-loss reconciled tree in normal form to the core problem. A similar dynamic programming approach for finding a smallest reconciled tree is followed for finding a dup-loss reconciled tree. In order to minimize the duplication and loss cost, it is necessary to know how to calculate losses, since $C_{\mathsf{d}}(R) = \sum_{v \in \mathsf{V}(R)} (k(v) - 1)$ for a reconciled tree R is clear.

Calculate Losses

Since losses can only be determined if embedding is known, to pre-computed losses, basically all possible embedding scenarios are considered by enumerating all layers in a complete subtree of a reconciled tree. As described in the recursion introduced in the previous chapter, only in the case of d-nodes, layers are embedded to complete subtrees in a reconciled tree. Otherwise, the recursion handles the subproblems.

Recall that in the events of d-nodes, all copies except one correspond to a complete subtree in S. The number of losses in such subtrees can be expressed by $\mathsf{Loss}(v, i)$, where $v \in \mathsf{V}(S)$ and $i \in \mathbb{N}_0$. Formally, $\mathsf{Loss}(v, i)$ is the number of losses in a subtree of R, where this particular subtree and S_v are isomorphic, with i-th layer of C_v embedded. It can be formulated as

$$\mathsf{Loss}(v,i) = \begin{cases} 0, & \text{if } 1 \le i \le n(v); \\ 1, & \text{if } i > m(s) \text{ or } i \le 0; \\ \sum_{u \in \mathsf{Ch}(v)} \mathsf{Loss}(u, i - n(s)), & \text{otherwise.} \end{cases}$$

The result of the above recursion can be pre-computed and stored in a lookup table of size $O(n^2)$.

Once the number of copies of each node is determined, calculating the total losses can be done in $O(n)$ steps in the resulting reconciled tree.

Let $c(C, S) = C_{\mathsf{d+l}}(R)$ where $R \in \mathsf{Rec}^2(C, S)$ under an embedding function Emb and a duplication Dup, $r = \mathsf{Root}(R)$, and $s = \mathsf{Root}(S)$. The value $c(C, S)$ can be calculated by a similar recursive approach used in Chapter 6 according to Lemma 5.6.

Case I: bases case, $n(s) = m(s)$ and $k = \max\{1, n(s)\}$. Hence r is an s-node if $k = 1$ and a d-node if $k > 1$.

$$c(C, S) = (k - 1) + \sum_{i=1}^{k} \mathsf{Loss}(s, i), \tag{7.1}$$

where $C_{\mathsf{d}}(R) = k - 1$.

Case II: recursive case, $n(s) < m(s)$ and $k(s) = k \geq 1$. Again, r is an s-node if $k = 1$ and a d-node if $k > 1$.

$$c(C, S) = \min_{k} \big\{(k-1) + \sum_{i=1}^{k-1} \textsc{Loss}(s, i) + \sum_{u \in \textsc{Ch}(s)} c(\textsc{Reman}(C_u, k-1-n(s)), S_u)\big\}, \qquad (7.2)$$

where $\max\{1, n(s)+1\} \leq k \leq m(s)$. The reasoning of the above recursion is exactly the same as (6.2).

Determine a Dup-Loss Reconciled Tree

Following the similar argument that concludes (6.3), a recursive function $\Gamma(v, i)$ can be designed to replace $c(\textsc{Reman}(C_v, i), S_v)$ such that $k(v, i)$ can be determined by using a dynamic programming approach.

$$\Gamma(v, i) = \begin{cases} (k_0 - 1) + \sum_{j=i+1}^{i+k_0} \textsc{Loss}(v, j), & \text{if } n(v) = m(v); \\ \min_{k_1 \leq k \leq k_2} \big\{(k-1) + \sum_{j=i+1}^{i+k-1} \textsc{Loss}(v, j) + \sum_{u \in \textsc{Ch}(v)} \Gamma(u, k-1-n'(v))\big\}, & \text{otherwise,} \end{cases} \qquad (7.3)$$

where $n'(v) = n(v) - i$ and constants k_0, k_1 and k_2 are the same as in (6.4).

Similar to the case of smallest reconciled trees, once $k(v, i)$ is determined, a dup-loss reconciled tree can be constructed in the exact same way by backtracking at $k(s) = k(s, 0)$. The above recursions can also be adapted to compute the duplication and loss cost of a smallest reconciled tree as a fitness measure between G and S suggested by Page [17].

8 DISCUSSION ON OPEN PROBLEMS

This chapter discusses possible future work relating to the gene duplication model. Especially important is work that may benefit from the application of the soft multifurcation hypothesis.

Enumerating All Reconciled Trees

As shown in previous examples, reconciled trees are not necessarily unique when multifurcation is involved. The number of different reconciled trees from a given gene tree to a given species tree could indicate the accuracy of a resulting reconciled tree.

Apparent Polytomies in Species Trees

The motivation of our extension to the original GD model is apparent polytomy which assumes that multifurcations in a gene tree can be replaced by a complete binary tree freely. One naturally adapts the strategy to replace multifurcations in the species tree which can be standalone or in conjunction with gene tree apparent polytomy. Our current conjecture is that such extension does not change the complexity of the GD model. Because of the embedding requirement in a reconciled tree, after proper binary species subtree replacements, the reconciliation result still depends on the embedding model. A formal analysis of the conjecture will be presented in our future work.

Multiple Gene Duplication

This problem can be considered as a variation of OST [8, 14], where a single duplication event affects all (multiple) gene trees, rather than just a single gene tree. In other words, one single reconciled tree is sought for every input gene tree instead of independent reconciled trees from each gene tree.

Gene Duplication Supertree

Since the gene duplication model provides a measure to compare a species tree and a gene tree, by giving a set of gene trees, it should be possible to find a species tree that *best fits* all of the gene trees. In that sense, the best fit species tree is an assembly of the input gene trees.

Note that the postulated result should only be considered as a species phylogeny while the reconciled tree(s) provide an explanation of possible gene phylogenies.

Deep Coalescence and Horizontal Gene Transfer

The concept of relaxed embedding is independent of the gene duplication model. It should be reasonable to adapt other tree mapping models for the soft multifurcation assumption which, as illustrated earlier, provides a better explanation for multifurcated input trees.

Performance Evaluation

One of the motivations behind the gene duplication model is to identify potential duplication events which support the model. Similarly, there are assumptions in this extended model that need to be evaluated as well. The dynamic programming solutions are reasonable to implement and real datasets can be used as experiments. A comparison between the two reconciled tree optimizations may yield interesting results since it is only known the two can be different.

APPENDIX A ADDITIONAL DEFINITIONS AND PROOFS

Common Definitions

The following definitions are common in literatures. To fit this work better and avoid ambiguity, there are subtle adaptations hence they are included.

Definition A.1 (graphs). A *directed graph* G is an ordered pair (V, E), where V is a finite, non-empty set named *vertex set* or *node set*, denoted by $\mathsf{V}(G)$, and $E \subseteq V \times V$ is named *edge set*, denoted by $\mathsf{E}(G)$. Elements of V are called *vertices* or *nodes*; elements of E are called *edges*. An edge $e \in E$ is called a *self-loop* if $e = (v, v)$, where $v \in V$, and G is *simple* if it has no self-loops.

An *undirected graph* $G = (V, E)$ is defined as a simple directed graph such that if $u, v \in V$ and $u \neq v$ then $(u, v) \in E \Leftrightarrow (v, u) \in E$. For simplicity, consider $(u, v) = (v, u)$ for undirected graphs. By converting a directed graph $G' = (V', E')$ into an undirected graph $G = (V = V', \{\, (u, v) : ((u, v) \in E' \vee (v, u) \in E') \wedge u \neq v \,\})$, G is the *underlying graph* of G'.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, G' is a *subgraph* of G. If $V' \subseteq V$ and $E' = \{\, (u, v) \in E : u, v \in V' \,\}$, G' is the subgraph of G *induced* by V', denoted by $G_{V'}$.

Let $G = (V, E)$ be a graph and $(u, v) \in E$. Then v is called *adjacent* to u. The set of vertices adjacent to u is denoted by $\mathsf{Adj}_G(u)$. If G is an undirected graph, then $\mathsf{Deg}_G(u) = \big|\{\, w \in V : (u, w) \in E \,\}\big|$ is defined as the *degree* of u. If G is a directed graph, $\mathsf{Deg}^-_G(u) = \big|\{\, w \in V : (w, u) \in E \,\}\big|$ is defined as the *in-degree* of u and $\mathsf{Deg}^+_G(u) = \big|\{\, w \in V : (u, w) \in E \,\}\big|$ as the *out-degree* of u.

A *path* p in a graph $G = (V, E)$ from u, $v \in V$ is a sequence $p = \langle v_0, v_1, v_2, \ldots, v_{k-1}, v_k \rangle$ of vertices in V such that $u = v_0$, $v = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1,\ 2,\ \ldots,\ k$. The *length* of p is the number of edges contained in the path, denoted by $\big|p\big|$; in this case, $\big|p\big| = k$. A path is *simple* if there is no repeated vertices in the sequence of vertices. Note that the shortest path length can be 0, and such path contains only 1 vertex.

A *cycle* c of length k in a graph G is a path $\langle v_0, v_1, v_2, \ldots, v_{k-1}, v_k \rangle$ in G such that $v_0 = v_k$. The cycle c is *simple* if there is no other repeated vertices except $v_0 = v_k$. Similar to the length of a path, the length of a cycle c is denoted by $\big|c\big|$. Note that a self-loop is a cycle of shortest length, which is 1, and it can only exist in directed graphs.

A graph G is *cyclic* if there is a cycle in G, otherwise, G is *acyclic*.

An undirected graph G is *connected* if there is always a path between any pair of its vertices. A directed graph is (weakly) connected if its underlying graph is connected.

Two graphs G and G' are *isomorphic* if there is a bijective (one-to-one correspondence) function $f : \mathsf{V}(G) \to \mathsf{V}(G')$, called a *graph isomorphism*, such that $(u, v) \in \mathsf{E}(G) \Leftrightarrow (f(u), f(v)) \in \mathsf{E}(G')$.

Definition A.2 (trees). A *(rooted phylogenetic) tree* T is a directed acyclic graph with the following properties:

- There is exactly one special node named *root*, denoted by $\mathsf{Root}(T)$, such that $\mathsf{Deg}^-_T(\mathsf{Root}(T)) = 0$, and $\mathsf{Deg}^-_T(u) = 1$ for all $u \in \mathsf{V}(T) - \{\mathsf{Root}(T)\}$.
- There is a subset of $\mathsf{V}(T)$ named *leaf set*, denoted by $\mathsf{Le}(T)$, such that $\mathsf{Deg}^+_T(v) = 0$ for all $v \in \mathsf{Le}(T)$, and $\mathsf{Deg}^+_T(u) \geq 2$ for all $u \in \mathsf{V}(T) - \mathsf{Le}(T)$.

An element in $\mathsf{Le}(T)$ is called a *leaf node*; an element in $\mathsf{V}(T) - \mathsf{Le}(T)$ is called an *internal node*. The set of all internal nodes in T is denoted by $\mathsf{In}(T)$.

Since the in-degree of a node v in a tree T is always at most 1, for simplicity, the out-degree of v is used as the degree of v in a tree, i.e. $\mathsf{Deg}_T(v) = \mathsf{Deg}^+_T(v)$.

Although phylogenetic trees can also be unrooted, i.e. there is no such special rooted node, only rooted phylogenetic trees are focused here. Thus a *tree* means a rooted phylogenetic tree unless stated otherwise.

The *size* of a tree T, denoted by $\big|T\big|$, is the cardinality of its vertex set, $\big|\,\mathsf{V}(T)\big|$.

A tree T is *binary* or *bifurcated* if every internal node of T has a degree of 2, otherwise T is *polytomy* or *multifurcated*. A *star-tree* is a tree with only its root node as internal node.

Let u and v be nodes in a tree T. If $v \in \mathsf{Adj}_T(u)$, the node u is called the *parent* of v, denoted by $\mathsf{Pa}_T(v)$, and v a *child* of u. The set of all children of u is denoted by $\mathsf{Ch}_T(u)$. If there is a path from u to v in T, denoted by $u \leq_T v$, u is called an *ancestor* of v and v a *descendant* of u; otherwise, $u \not\leq_T v$ denotes that there is no path from u to v in T. If $u \leq_T v$ and $u \neq v$, denoted by $u <_T v$, u is called a *proper ancestor* of v and v a *proper descendant* of u. For convenience, $\mathsf{Pa}^*_T(v)$ denotes the set of all ancestors of v; $\mathsf{Ch}^*_T(v)$ denotes the set of all descendants of v.

The *depth* of a node u in a tree T, denoted by $\mathsf{Depth}_T(u)$, is the length of the path from the root of T to u.

Let X be a subset of $\mathsf{V}(T)$ in a tree T. With the relation $\leq_T$, $(X, \leq_T)$ defines a partially ordered set. An element $v \in X$ is called a *minimal element* of X if there is no element $u \in X$ such that $u \leq_T v$, and v is a *maximal element* of X if there is no element $u \in X$ such that $v \leq_T u$. The set of all minimal elements of X is denoted by $\min_T(X)$, and the set of all maximal elements of X is denoted by $\max_T(X)$. Formally,

$$\max_T(X) = \{x \in X \mid \nexists y \in X \wedge x \leq_T y\};$$
$$\min_T(X) = \{x \in X \mid \nexists y \in X \wedge y \leq_T x\}.$$

Definition A.3 (complete subtrees). The (*complete*) *subtree of* T *rooted at* u, where $u \in \mathsf{V}(T)$, is the subgraph of T induced by $\mathsf{Ch}^*(u)$ and denoted by T_u .

Additional Proofs

The collection of proofs to lemmata described in earlier chapters.

Lemma A.4 (3.16).

Proof. (by contradiction)

Let T be an embedding tree of G under an embedding function Emb and $g \in \mathsf{In}(G)$.

Assume that for any two distinct children c_1, c_2 of g, $\mathsf{Emb}(g) < \mathsf{LCA}(\{\mathsf{Emb}(c_1), \mathsf{Emb}(c_2)\})$. Then there exists exactly one child node x of $\mathsf{Emb}(g)$ such that $\mathsf{Emb}(g) < x \leq \mathsf{LCA}(\{\mathsf{Emb}(c_1), \mathsf{Emb}(c_2)\})$,

otherwise there is bound to exist two children c_3, c_4 of g such that $\mathsf{Emb}(g) = \mathsf{LCA}(\{\mathsf{Emb}(c_3), \mathsf{Emb}(c_4)\})$. It follows that $\mathsf{Emb}(\mathsf{Ch}(g))$ is a subset of $\mathsf{V}(T_x)$ and $\mathsf{Emb}(g) \notin \mathsf{V}(T_x)$.

Specifically, $\mathsf{Emb}(g)$ is not a node in any restriction of T_x. This implies that Emb can not be an embedding function from G to T. □

Lemma A.5 (3.17).

Proof. (by construction)

Let G and S be trees with a leaf association A. Without loss of generality, let $n = |\mathsf{Le}(G)|$, and we label each node in $\mathsf{Le}(G)$ as $l_1, l_2, \ldots, l_n$. Construct a tree T using the following steps, and assume that all the node sets in the following trees are disjoint, except in T.

1. Construct a tree T_0 isomorphic to G where f_0 is an isomorphism.
2. Construct n trees $T_1, \ldots, T_n$ isomorphic to S such that f_i : is an isomorphism for $1 \leq i \leq n$.
3. Let $\mathsf{Root}(T) = \mathsf{Root}(T_0)$, and replace each node $v \in \mathsf{Le}(T)$ by $\mathsf{Root}(T_i)$ for $1 \leq i \leq n$.
4. The embedding function:

$$\mathsf{Emb}(v) = \begin{cases} f_0(v), & \text{if } v \in \mathsf{In}(G); \\ f_i \circ \mathsf{A}(v), & \text{if } v \in \{l_1, l_2, \ldots, l_n\}. \end{cases}$$

5. The duplication function:

$$\mathsf{Dup}(v) = \begin{cases} \mathsf{Root}(S), & \text{if } v \in \mathsf{V}(T_0)\ ; \\ f_i(v), & \text{if } v \in \mathsf{V}(T_i)\ . \end{cases}$$

With Emb and Dup, T is an explanation tree from G to S, as illustrated by Figure A.1.

□

Lemma A.6 (3.18).

Proof. (direct proof)

From Emb, it holds that $u \leq v \Rightarrow \mathsf{Emb}(u) \leq \mathsf{Emb}(v)$, since Emb is an isomorphism. It does not matter whether Emb is a strict embedding or a relaxed embedding function.

For any edge (w, z) on the path form $\mathsf{Emb}(u)$ to $\mathsf{Emb}(v)$, either $(\mathsf{Dup}(w), \mathsf{Dup}(z))$ is an edge in S or $\mathsf{Dup}(w) = \mathsf{Dup}(z)$. In either case, $\mathsf{Dup}(w) \leq \mathsf{Dup}(z)$ holds. Combining all edges from $\mathsf{Emb}(u)$ to $\mathsf{Emb}(v)$, it holds that $\mathsf{Dup} \circ \mathsf{Emb}(u) \leq \mathsf{Dup} \circ \mathsf{Emb}(v)$.

Hence $u \leq v \Rightarrow u'' \leq v''$. □

Corollary A.7 (3.19).

Proof. (by induction from leaf nodes to their ancestors)

Let u be a node in $\mathsf{V}(G)$. The base case is that u is a leaf node, and the inductive case is that u is an internal node. For convenience, if $v \in \mathsf{V}(G)$, then $u'' = \mathsf{Dup} \circ \mathsf{Emb}(v)$.

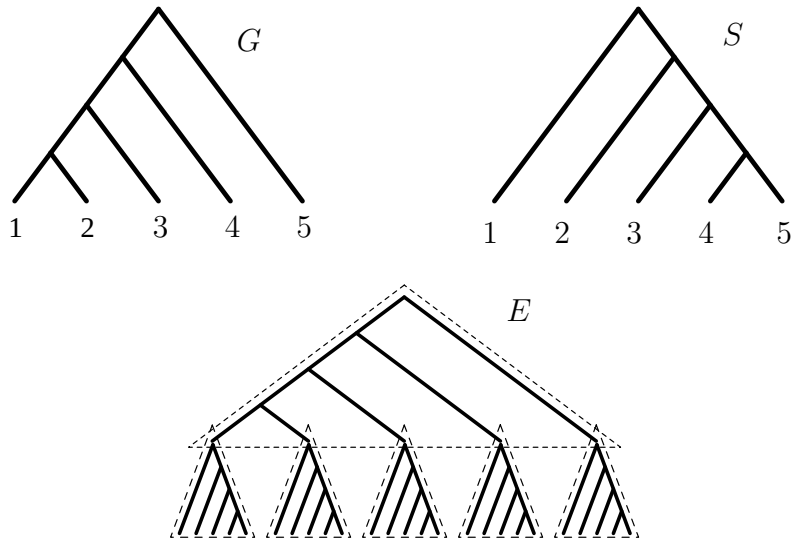

Figure A.1 The explanation tree E based on G and S that can be constructed universally.

Base step: u is a leaf

By the definitions of LCA mapping, duplication and embedding functions, $\mathsf{LCA}(u) = u''$.

Inductive step: u is an internal node

Assume that any child $v \in \mathsf{Ch}_G(u)$, $v'' \leq \mathsf{LCA}(v)$. It can be argued that $u'' \leq \mathsf{LCA}(u)$. Because of $u < v$ and Lemma 3.18, $u'' \leq v'' \leq \mathsf{LCA}(v)$. In other words, u'' is a common ancestor of $\mathsf{LCA}(\mathsf{Ch}(u))$. Hence $u'' \leq \mathsf{LCA}(u)$ otherwise it contradicts Definition 3.3.

□

APPENDIX B NOTATIONS

$\mathbb{N}_0$	The set of natural numbers, i.e. $\{0, 1, 2, \ldots\}$.
$\mathbb{Z}^+$	The set of positive integers, i.e. $\{1, 2, 3, \ldots\}$.
$\lvert T \rvert$	The cardinality of a tree T.
$\mathsf{V}(T)$	The vertex (node) set of a tree T.
$\mathsf{E}(T)$	The edge set of a tree T.
$\mathsf{Root}(T)$	The root node of a tree T.
$\mathsf{Le}(T)$	The leaf set of a tree T.
$\mathsf{In}(T)$	The set of internal nodes of a tree T.
$u \leq_T v$	There is a path from u to v in tree T, where u, $v \in \mathsf{V}(T)$.
$u <_T v$	There exists a path from u to v in tree T where u, $v \in \mathsf{V}(T)$ and $u \neq v$.
$\max_T(X)$	The set of all maximal elements of $X \subseteq \mathsf{V}(T)$.
$\min_T(X)$	The set of all minimal elements of $X \subseteq \mathsf{V}(T)$.
T_v	The complete subtree T rooted at a node $v \in \mathsf{V}(T)$.
$\mathsf{Depth}_T(v)$	The depth of a node v in a tree T.
$\mathsf{Pa}_T(v)$	The parent of a node v in a tree T.
$\mathsf{Ch}_T(v)$	The set of children of a node v in a tree T.
$\mathsf{Pa}^*_T(v)$	The set of all ancestors of a node v in a tree T.
$\mathsf{Pa}^+_T(v)$	The set of all proper ancestors of a node v in a tree T.
$\mathsf{Ch}^*_T(v)$	The set of all descendants of a node v in a tree T.
$\mathsf{Ch}^+_T(v)$	The set of all proper descendants of a node v in a tree T.
LCA_A	A LCA mapping function with leaf association A.
$T_{\lvert X}$	The restriction of T to X.
$T' \leq T$	A tree T is a refinement of a tree T'.
$C_\mathsf{d}(R)$	The duplication cost of R.
$C_\mathsf{l}(R)$	The total losses of R.
$C_{\mathsf{d}+\mathsf{l}}(R)$	The duplication and losses cost of R.
$\mathsf{Exp}_A(G, S)$	All explanation trees from G to S with leaf association A.
$\mathsf{Exp}^*_A(G, S)$	All special explanation trees from G to S with leaf association A.
$\mathsf{Rec}_\mathsf{A}(G, S)$	All reconciled trees from G to S with leaf association A.
$\mathsf{Rec}_{\mathsf{LCA}}(C, S)$	All reconciled trees in the core problem.
$T^{(g)}$	The cut-out tree of T by $g \in \mathsf{V}(T')$, where T is an embedding tree of T'.
GDP	The Gene Duplication Problem.
C_v	The subset of C mapped into the subtree rooted at v (depending on which tree v belongs to).
$\textsc{Top}(X)$	The set of all top elements of X.
$\textsc{Layer}(X, i)$	The i-th layer of X.
$\textsc{Remain}(X, i)$	The i-th remain of X.

$n(v)$	The number of non-empty layers of C_v across the node v.
$m(v)$	The number of non-empty layers of C_v.

BIBLIOGRAPHY

[1] M. A. Bender and M. Farach-Colton. The lca problem revisited. *Latin American Theoretical Informatics*, pages 88–94, 2000. Apr.

[2] D. Bryant. A classification of consensus methods for phylogenetics. In *DIMACS-AMS*, pages 163–184, 2003.

[3] W. Chang. Gene tree reconciliation with soft multifurcations. Master's thesis, Iowa State University, August 2005. (To be available as a technical report or upon request.).

[4] K. Chen, D. Durand, and M. Farach-Colton. Notung: Dating gene duplications using gene family trees. In *RECOMB*, pages 96–106, 2000.

[5] T. H. Cormen, C. E. Leiserson, R. L. Riveset, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge-London, second edition, 2001.

[6] O. Eulenstein. A linear time algorithm for tree mapping, 1997. `http://taxonomy.zoology.gla.ac.uk/rod/genetree/maths/maths.html`, (access date: February 19, 2006), Arbeitspapire der GMD, 1046.

[7] O. Eulenstein. *Vorhersage von Genduplikationen und deren Entwicklung in der Evolution*. Ph.d. dissertation, Bonn University, 1998. `http://www.bi.fraunhofer.de/publications/research/1998/020/Text.pdf`, GMD Research Series, 20.

[8] M. Fellows, M. T. Hallett, and U. Stege. On the multiple gene duplication problem. In *ISAAC: 9th International Symposium on Algorithms and Computation*, pages 347–356, 1998.

[9] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

[10] M. Goodman, J. Czelusniak, G. W. Moore, A. E. Romero-Herrera, and G. Matsuda. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology*, 28:132–163, 1979.

[11] R. Guigó, I. Muchnik, and T. F. Smith. Reconstruction of ancient molecular phylogeny. *Molecular Phylogenetics and Evolution*, 6(2):198–213, 1996.

[12] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York-Melbourne, 1997.

[13] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[14] B. Ma, M. Li, and L. Zhang. On reconstructing species trees from gene trees in term of duplications and losses. In *RECOMB*, pages 182–191, 1998.

[15] W. P. Maddison. Reconstructing character evolution on polytomous cladgrams. *Cladistics*, 5:365–377, 1989.

[16] B. Mirkin, I. Muchnik, and T. F. Smith. A biologically meaningful model for comparing molecular phylogenies. *Journal of Computational Biology*, 2:493–507, 1995.

[17] R. D. M. Page. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology*, 43(1):58–77, 1994.

[18] R. D. M. Page and E. C. Holmes. *Molecular Evolution: A Phylogenetic Approach*. Blackwell Science Ltd, Osney Mead, Oxford, 1998.

[19] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[20] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, Oxford-New York, 2003.

[21] J. B. Slowinski. Molecular polytomies. *Molecular Phylogenetics and Evolution*, 19(1):114–120, 2001.

[22] L. Zhang. On a mirkin-muchnik-smith conjecture for comparing molecular phylogenies. *Journal of Computational Biology*, 4(2):177–187, 1997.

[23] C. M. Zmasek and S. R. Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, 17(9):821–828, 2001.